Emergent Communication for a Hidden Role Game: The Resistance

Chih-Yu Lai
chihyul@mit.edu

Eliot Cowan
CaptainFantastic@mit.edu

Abstract

We trained proximal policy optimization (PPO) agents to play the hidden role game "The Resistance". Learning whether or not other actors are behaving in your interest, or only pretending to, is a problem widely unstudied in reinforcement learning. We allow the agents to create and develop their own form of communication which allows them to adversarially influence the actions of other agents. We develop several baseline strategies and metrics to evaluate and quantify our training results. A total of 10 models are constructed and used for completing different tasks during the game by two competing teams. We found that the PPO agents can play competitively against our baseline strategies, without training on these baselines. This means the agents not only learn to play against their non-stationary counterparts, but learn generic strategies to play against unknown players. Our experimental results show that the agents developed communication in order to identify each other's roles, resulting in an increase of their win rates. Therefore, we've shown that emergent communication is helpful for cooperative and adversarial multi-agent reinforcement learning when there are partially observable states.

1 Introduction

Hidden roles games like Secret Hitler, Mafia, Avalon, or the Resistance have remained largely unsolved by the RL community [1]. As a player, these games require you to determine who is on your team and trick other players into thinking you're on their team. Existing research presents initial approaches to solving such games, but none of them allow communication between the actors: a critical part of the social deduction process to reveal players' true [2]. Communication is essential during game-play to collect, share and obfuscate information between players. Based on new literature in the field of Emergent Communication [3, 4, 5, 6], we present the first approach to The Resistance where agents can communicate with each other.

For the resistance (a version of Avalon), there have previously been strong results using counterfactual regret minimization and deep value networks; however, these results do not incorporate communication into the game-play [1]. In addition, PPO has been successful in solving many other multi-agent games [7], so we pursued the development of PPO agents capable of playing and solving The Resistance with communication.

First, we provide a formal definition of The Resistance, the game environment and the agents' models. Then, we construct the multi-agent reinforcement learning training cycle which helps the agents play and improve against each other during training. Finally, we evaluate the agents with and without communication using several metrics designed for these experiments.

1.1 Game Rules

The Resistance is a 5-player game where each player is randomly assigned a team. There are two teams: the evil team, "Minions of Mordred," and the good team, "Loyal Servants of Arthur." The

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

number of players on each team varies with the number of players in the game. With 5 players, there are 3 good players and 2 evil players. Since this is a game of social deduction, the members of the blue (good) team don't know who else is on their team. The red (evil) team members start the game knowing the identities of all players.

One round of game play consists of a single player (the leader) proposing a group of people to go on a mission together. The mission composition is simultaneously voted on by every player in the game; if there is not a majority agreement then the mission fails. If 5 consecutive rounds fail, the red team wins immediately. However, if the mission vote succeeds, then the players on the mission are given 2 cards and secretly choose one of them: "succeed" or "sabotage." A blue team member must always choose to make a mission succeed. Evil team members are the only ones who may choose to (in secret) sabotage a mission. Each mission only requires one sabotage card to be chosen to make the mission a failure. Once a round ends, the leader moves to the next clockwise player and the cycle starts over. **The evil team's goal is to sabotage 3 missions out of the 5 total missions. Similarly, the good team's goal is to have 3 successful missions out of the 5 total missions. The good team members need to figure out who is on their team so they can stop the evil players from sabotaging too many missions.** The number of players on each mission changes as the game goes on. The five missions during the game have size 2, 3, 2, 3, and 3 respectively.

1.2 Emergent Communication

The field of emergent communication has been rapidly developing within the past five years [4, 5, 8]. Most of the research has focused on allowing learned communication between agents for the purpose of cooperation between (specialized) agents in multi-agent systems [9]. Each player's communication is treated as a continuous action, completely determined by a parameterized reinforcement learning model. The output of this model is then inputted to all other players for them to use when deciding how to act. These methods have primarily been used in cooperative games where, for example, one "manager" agent is given a picture of blocks and must inform another "builder" agent what to do in order to build that structure. We applied these methods to aid cooperation between agents, but novelly **we also used these emergent communication methods to let agents trick one another** in the context of The Resistance.

2 Method / Problem Formulation

2.1 Experimental

The computational requirements were high since multiple DNNs needed to be trained by PPO. A NVIDIA GeForce 3090X GPU was used alongside an AMD Ryzen 9 5950X 16-Core Processor and Two 32GB DDR4 DRAMs running at 2666MHz. We used PyTorch as the training framework.

2.2 Problem Formulation

Below we formally introduce naming conventions for the values, networks, and training components used during game play. Then we define the state space, action space, objective function, and algorithm used for each model. See Tables 1, 2, and 3 which collectively define the naming conventions for the game's variables. Each player must complete up to five different tasks. See Table 4 for the description of each of these tasks as well the their input and action spaces.

We define all the models' state spaces to encompass all previous public actions (except prior communications) in order to preserve the Markov assumption. Therefore, the state after a single round consists of a vector indicating which player was the leader, which players the leader chose to go on the mission, voting results, mission results, the cumulative mission failures and the round number. These have sizes 5, 5, 5, 3, 1, and 1 respectively, resulting in a 20 dimensional flattened vector. Since there are up to 5 total missions and since the game ends if there are ever more than 4 consecutive "no" votes to mission participants, there are up to 25 total rounds in the game. This means we need up to $20 * 25 = 500$ values to represent the complete public history of a game. However, since we have a fixed input shape, we must provide an input to the model at each of these locations. If a round

hasn't happened yet, we must initialize the parameters associated with that round to 0, but to ensure that the model doesn't confuse the initial value of this state with a round that occurred without any voting (or going on any mission) we must add an additional indicator bit to the input space for each existing input that informs the model whether or not the associated bit is part of a round that has actually occurred or just padding. This means we have a total input space of size $2 * 500 = 1000$ values. The tables below (Tables 1, 2, 3) describe how we constructed this history vector $hist$.

Table 1: Naming conventions for scalars.

Name	Symbol	Description
blue team id	0	
red team id	1	
current mission id	mi	$0 \leq mi < 5$
current round id	ri	$0 \leq ri < 5$
number of rounds in mission mi	$nr[mi]$	
current step id	si	$si = \sum_{i=0}^{mi} nr[i]$
current leader id	li	$0 \leq li < 5$
player id	pi	$0 \leq pi < 5$
current number of consecutive fails	nf	$0 \leq nf \leq 3$

Table 2: Naming conventions for vectors.

Name	Symbol	Description
number of players going on mission	nm	$nm = [2, 3, 2, 3, 3]$
ground truth player identities	$every$	$every[pi] \in [0, 1]$ and $\sum_{i=0}^{4} every[i] = 2$
predicted player identities	who	$0 \leq who[pi] \leq 1$
self one-hot vector	$self$	$self[pi] = 1$ if pi is current player, else 0
leader one-hot vector	$lead$	$lead[pi] = 1$ if pi is current leader, else 0
players on mission vector	$miss$	$miss[pi] = 1$ if pi selected on mission, else 0
vote vector	$vote$	$vote[pi] = 1$ if player pi votes yes, else 0
success vector	$succ$	$succ[i] = 1$ if player i succeeded the mission, else 0

Table 3: Naming conventions for matrices.

Name	Symbol	Description
communication matrix	$comm$	A $5 \times N_{comm}$ matrix that records what each agent expresses at the start of each round.
all state information	$hist$	A $2 \times 25 \times 20$ matrix that record all current available information from game start.

2.3 Trainable Models

At the beginning of the game, two agents are initialized (one red, one blue) that each have their own trainable parameters for the tasks (described in Table 4). Each time, for example, a red player needs to complete a task, the history available to that player is inputted into the red agent's respective model for that task and an action is sampled from the returned distribution.

All the models that use PPO for training have a similar neural network backbone structure: a four-layer dense network with input dimensions 256, 64, 32, and 64. The layers are separated by Rectified Linear Units (ReLU). A final layer connects the backbone structure to the output, which differs in size for every task. The output is then used to create a categorical or continuous distribution from which an action is sampled. [1]

[1] For the *WHO* network, a dense network of three hidden layers with 256, 64, and 32 features respectively is used, followed by an output layer with an output dimension of 5. The output doesn't connect to a distribution.

Table 4: Models used for different tasks during one round. (See Tables 1, 2, and 3 for a full description of each variables.)

Network name	Input (State)	Output (Actions)	Description
$COMM$	$self, who/every, hist$	$comm[pi]$	
$MISSION$	$self, who/every, hist$	$miss$	
$VOTE$	$self, who/every, hist$	$vote[pi]$	li and $miss$ in the same round is included in $hist$
$SUCCESS$	$self, who/every, hist$	$succ[i]$	$vote$, li and $miss$ in the same round is included in $hist$
WHO	$self, comm, hist$	who	

The *COMM* network is used at the start of a round when each player tries to **communicate** with each other. We concatenate the communication vectors from each of the 5 players to form $comm$, a $5 \times N_{comm}$ matrix. $comm$ is inputted into the WHO model (as defined in Table 4) which is used by the blue players to decide who their friends are. In theory, the players on the red team need to output a communication vector that tricks the blue team into believing they are also on the blue team.

The *MISSION* network is used by the current leader of the round for selecting $nr[mi]$ candidate players to go on a mission. It outputs a 1×10 vector, where each index corresponds to a specific combination for choosing $nr[mi]$ players from the total of five players. Generally, if the leader of a mission is on the blue team, we suspect that player may wish to choose the $nr[mi]$ players that have the minimum who values (since blue team id = 0) to go on their mission. Or they can try different strategies such as testing if some player is on the red team or not by choosing them. The overall goal for this model is to select a group of candidates that all the players will most likely approve, while still producing a mission that helps the leader win the game.

The *VOTE* network is used by every player to determine whether or not to approve a set of candidate players for a mission. It takes in the full public history from previous rounds ($hist$), the leader of the round, the selected candidate players, and its own estimate of who is on which team. If the majority of players voted yes, then the candidates go on mission; else another round starts and the next player becomes leader. For the blue team, a general strategy is to approve the mission if the agent thinks that all the candidates are blue teams, and disapprove the mission if it thinks any of them are red. However, since the red team immediately wins if there are five consecutive rounds without a majority of approval votes, the blue agents need to balance the trade-offs between being too conservative and over-optimistic.

The *SUCCESS* network is only used by the red team, since all the players on the blue team must choose to succeed their missions. The players on the red team may not always choose to sabotage the mission, since there is a trade-off between taking a step towards winning and not being suspected as a red team player.

The *WHO* network is used by the blue team for determining the identity of all other players. It returns a size 5 vector with values from 0 to 1 that represent each of the 5 players' probability of being on the red team. Naturally, each of the blue players know that it is on the blue team, so the value associated with itself is manually set to 0 whenever a player queries this model. If this model had perfect accuracy, the blue team members should always stop any mission from occurring that included a red team member, and win almost every game. The red team doesn't use this model since red players already know the identity of each player.

2.4 Training Flow

In each training epoch, we loop over each of the trainable models for an agent and freeze the parameters of all other models. An episode is defined as a full game where each time step begins just before the point where the actively training model needs to make a prediction. Each epoch, experiences in 50 episodes are gathered and used for training each model. For PPO training, we set the target KL divergence to 0.01, the clip ratio to 0.2, and the entropy coefficient to 0.01. If the model being trained belongs to the blue team, we provide a reward of 1 at the end of the episode if the blue team wins and provide a reward of -1 if the red team wins. The same applies to red team, respectively.

We calculate the discounted reward propagated back from the end of each episode with a discounted factor of $\gamma = 0.99$.

The task for the *COMM, MISSION, VOTE*, and *SUCCESS* networks are action-based, so we implemented PPO algorithms for training and provided rewards according to the outcome of each game. However the *WHO* network is actually a supervised learning problem since the ground truth labels (the real identities of the agents) are known while training. As such, we use binary cross-entropy to calculate the error between the predicted player identities and the ground truth (saved in the $every$ vector) at each time-step.

Finally, we use a multiplexing trick for speeding up the training process. By initializing 50 different environments and concatenating all their inputs, we are able to step through each round and collect the experiences simultaneously for all the episodes. Overall, this results in a $5\times$ faster training speed.

2.5 Badness Score

For the purpose of evaluation, it's helpful to have a heuristic that indicates how much an actor appears to be on the red team using only public information. We chose to use a metric that we coined the "badness score" which deterministically assigns each player a number between 0 and 1 solely based on the public actions they've made so far in a game. The higher the number, the more likely we believe that player is on the red team. For each player i it is calculated as:

$$\text{Badness Score} = \frac{\text{Number of sabotaged missions including player i}}{\text{Total number of missions including player i}} \tag{1}$$

If a player has never been included on a mission, the score is set to $\frac{1}{2}$ because the metric is meant to be used by a blue player who already knows that they are on the blue team. Therefore if that blue player chooses another player randomly from the remaining 4 players there is a $\frac{1}{2}$ chance that the chosen player is on the red team.

2.6 Baseline Agents

In order to evaluate agent performance, we created strong (mostly greedy) baseline agents for the red and blue team. For the red team, the baseline agent always elects itself to go on a mission and then selects additional players randomly. It votes yes on a mission *iff* there is at least one red player on the mission. Also, the baseline agent always sabotages a mission when given the opportunity.

For the blue team, the baseline agent will presume the red players are the two players who have the highest badness score (excluding itself). It settles ties randomly. Then, it presumes the blue agents are itself and the other two players that are not presumed to be on the red team. The baseline agent will always select itself and its presumed blue players to go on a mission. It votes yes on a mission if there are no presumed red players on the mission or if there have been four failed rounds previously (so another failure would cause the blue team to immediately lose). And, of course, it never sabotages a mission.

3 Results

3.1 Untrained Agent vs. Baseline

Prior to training, we evaluated our baselines against the untrained agents. In Table 5a, we see that the red baseline wins 100% of games when playing with an untrained blue agent, while the blue baseline wins 61% (Table 5b) of games when playing with an untrained red agent. This confirmed our suspicion that the red team begins with an advantage over the blue team.

3.2 Training Through Self-Play

After establishing untrained performance against the baselines, we trained the red and blue agent against each other using the training flow described in section 2.5.

Table 5: Results values from different experiments.

ID	Condition	Red win rate	Blue choice correctness	Vote yes portion - red	Vote yes portion - blue
5a	Untrained blue agent vs. red baseline	1.00	0.20	-	-
5b	Blue baseline vs. untrained red agent	0.39	0.37	-	-
5c	Trained blue agent vs. red baseline	**0.51**	0.61	-	-
5d	No Comm: Trained blue agent vs. red baseline	1.00	0.16	0.91	1.00
5e	Blue baseline vs. red baseline	0.20	-	0.50	0.73
5f	No Comm: Blue baseline vs. trained red agent	0.61	0.36	0.51	0.58
5g	Blue baseline vs. trained red agent	**0.64**	0.38	-	-

The red agent immediately learned to sabotage each missions and achieved a 100% success rate. So, we switched to only training the blue agent's models. After 500,000 episodes of training, it was clear the blue agent was not learning.

After some investigation, we determined that the blue agent was stagnating because the WHO model was unable to differentiate between red and blue agents. We tried training the WHO model for 1 million games while holding all other model parameters constant, but saw only marginal improvements in results. The best binary cross-entropy loss achievable (averaged over each of the 4 other players) was 0.31. The loss was lower bounded here until we modified the input to the WHO model to include the badness score for each player. This caused the loss to quickly jump down to 0.1. The blue agent performance quickly rose. In Fig. 1 we show the red model success rate over time where the blue agent WHO model receives the badness score of each player as an input.

As desired, the agent performance had rises and falls as the red and blue agents learned to exploit weakness in each other and adapt to new strategies. We were concerned that the WHO model might be returning the badness score with only a trivial transformation applied, so we tried replacing the WHO model with a deterministic function that always returned the badness score for each agent. After training the red and blue agents for one million games, the blue agent with the deterministic badness function performed about 20% worse than the blue agent with the deep WHO function that accepted the badness score as input. Therefore, **for all future experiments, we included the badness score as an input to the WHO models.**

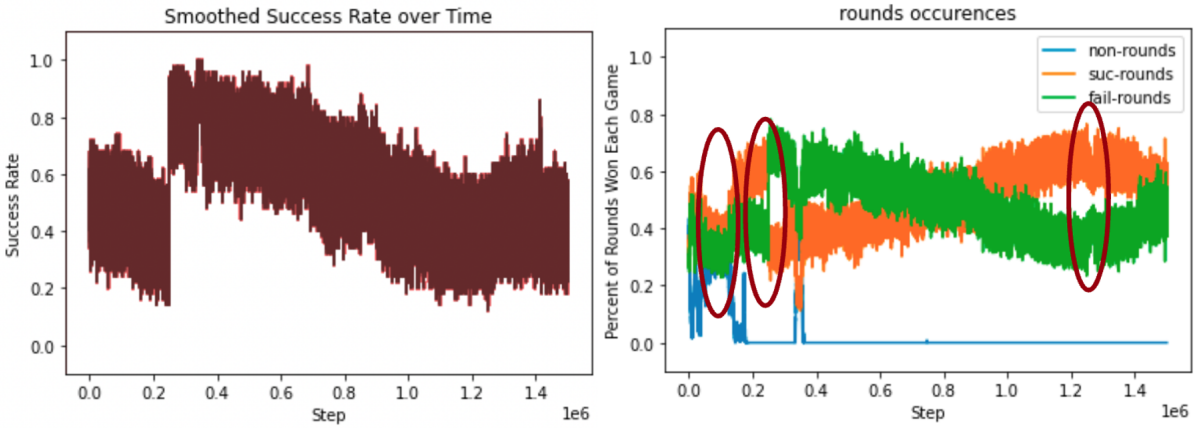

Figure 1: Results for red and blue agent training back-and-forth. Left: smoothed red win rate; right: Percent of rounds resolved by method. The training cycle switch points are circles in red.

3.3 Training Without Communication

When training without communication, we found an extremely low sample efficiency for the blue agent. After one million games, the blue agent still had not reached a 60% win rate against the untrained red agent. The binary cross-entropy for the WHO loss was around 0.4, which indicated that the blue agents were unable to discern their allies from their enemies. The results of the blue

agent against the baseline were comparable to the results of the untrained blue agent against same the baseline (Table 5d). We found that the blue agents were unable to coordinate their actions, which meant that the red agents were able achieve extremely high performance.

After five million games, the blue agent achieved good performance twice against the red agent before the red agent adapted. However, since the blue agents could not coordinate or communicate, they still achieved a 0% success rate against the red baseline. The red agent, on the other hand, had no trouble operating in the game undetected. It achieved competitive performance against the baseline agent with a 61% success rate. The results for this experiment are written in Table 5d and 5f. As such, we've shown that communication provides a comparative advantage for the blue team and is necessary for proper training of the blue agent.

3.4 Training With Communication

After allowing communication, we trained fresh red and blue agents against each other. The red agent achieved a win rate of 64% (Table 5g) against the blue baseline, whereas the untrained red agent only has a 39% win rate (Table 5b). This is a minor improvement compared to the red agent trained without communication.

Next, we evaluated the trained blue agent on the red baseline. The blue agent achieved a win rate of 0.49% (Table 5c), whereas the untrained blue agent had a win rate of 0% (Table 5a). We also found that blue players correctly chose their teammates 61% as opposed to 20% for the random agent. This means the blue baseline successfully leverages the *WHO* network, player's communications, and the history of the game to accurately predict who its allies are.

3.5 Baselines vs. Baselines and Other Experiments

When allowing the baselines to compete against each other, we found that the red baselines wins around 85% of the time (Table 5). We believed this was due to a limitation of the badness score, the metric the blue agents used to determine which players are enemies. We tried to improve the blue baseline's performance by adding a trainable *WHO* network in the place of the badness score. We found that the blue baseline performance drastically improved and could eventually win almost 100% of the time using the learned *WHO* function (Fig. 2). This confirmed the limitations of the badness score and confirmed that the *WHO* network is properly learning. Since we were training against the red baseline agent, we had no way to evaluate these models against the fully trainable agents.

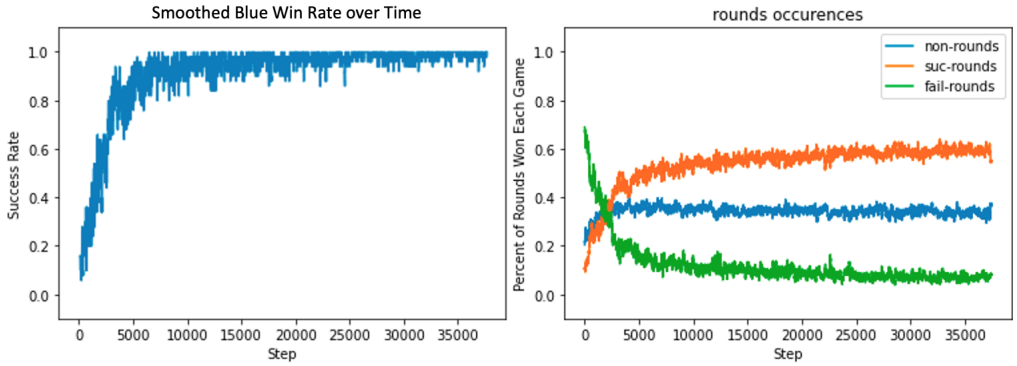

Figure 2: Training results for red baseline vs blue baseline using a *WHO* network. Left: smoothed blue win rate over time; right: winning conditions.

4 Unexpected Issues

Surprisingly, we found that using a deep value network to learn the value function for each task hurt performance. It resulted in a large reduction in the sample efficiency, so much that it was deemed infeasible to continue training. Since the objective function of each agents' models already changed so rapidly during training, we believe the additional instability added to the objective function by introducing deep value networks resulted in a very difficult and quickly changing reward landscape.

5 Conclusion & Discussion

We successfully trained the first RL agents for The Resistance. Our agents played competitively against strong, greedy strategies and altered their tactics to adapt to weaknesses in their opponents' strategies. Finally, we showed that communication was essential for creating high performance agents to play the Resistance.

We noticed a common pattern in the behaviors of the agents throughout the experiments. Generally the blue agent started off winning most games. Then, the red agent usually received an instant boost in performance near the beginning of the training process when it learned to always sabotage missions. At some point, the blue agent learned to avoid selecting the agents that have previously been on sabotaged missions. In response, the red agents lost many games until they learned that they could also win the game by forcing 5 consecutive rounds to pass without a mission. So, they began voting "no" to almost every mission. Eventually, the blue agents learned to detect that agents who always voted "no" on missions are red, so the blue performance rose again. Finally, the red agents responded by becoming more conservative about when they were willing to vote no. After this point, we were generally unable to discern the learned behaviours from the agents, but evaluation performance continued to rise.

The agents did not reach a >95% win rate against the baseline strategies. These baselines were designed to be as strong as possible for a deterministic strategy. Since the game involves some randomness, we think it may be impossible (or at least very challenging) to achieve a >95% win rate against a strong strategy like the ones employed by the baselines. However, there are a large set of improvements that could be made to raise performance. Below we suggest a set of future experiments which we expect will further improve agent performance.

1) Save a history of old red and blue agents to diversify training policies. Allowing agents to play alongside old versions of themselves and against older opponent agents has been shown in existing literature to provide greater training stability and improve performance [10]. 2) Each of our models received, as input, the full history of the game. We believe using RNNs and only inputting updates to the game history would improve performance further. 3) Switching from PPO to multi-agent DDPG is probably the most likely alteration to produce an increase to performance. It's a much more robust and well-studied training algorithm for multi-agent problems [10]. 4) If additional compute is available, it may be helpful to to reincorporate deep value functions into the models and train agents asynchronously to overcome the decreased sample efficiency.

To better determine the value of communication, we propose adding special characters to the game. One of the 3 blue agents would be chosen to be granted the title of "Merlin". This agent would be given the true roles of each character and must communicate these to the other blue players. The game can be extended even more to the game of Avalon, where one of the red team members is granted the special power of the "assassin". This role is the same as the standard red role, but if the assassin is correctly able to identify which of the red players is Merlin at the end of the game, then the red team instantly wins. This significantly complicates the communication landscape for the blue agents. Merlin is incentivized to share the identity of the red players with the blue players, but if he is "too obvious", then the assassin will identify him and win the game. This sets up the agents for a much more complicated transfer of information and is as a relatively simple modification for future research using our open source Resistance gym: `https://github.com/eli4224/avalon/blob/main/main.ipynb`.

6 Contributions

We did all our proposals, reports, and coding together. Specifically, Eliot focuses on the PPO algorithm and evaluation metrics, and Chih-Yu focuses on the environment and engine.

References

[1] Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. Finding friend and foe in multi-agent games. *Advances in Neural Information Processing Systems*, 32, 2019.

[2] Jack Reinhardt. Competing in a complex hidden role game with information set monte carlo tree search. *arXiv preprint arXiv:2005.07156*, 2020.

[3] Jakob Foerster, Yannis Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv:1605.06676*, 2016.

[4] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. *arXiv preprint arXiv:1612.07182*, 2016.

[5] Paul Pu Liang, Jeffrey Chen, Ruslan Salakhutdinov, Louis-Philippe Morency, and Satwik Kottur. On emergent communication in competitive multi-agent teams. *arXiv preprint arXiv:2003.01848*, 2020.

[6] Jesse et. al. Mu. Emergent communication of generalizations. *arXiv:2106.02668*, 2021.

[7] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[8] Ryan Lowe. On the pitfalls of measuring emergent communication. *arXiv:1903.05168*, 2019.

[9] Antoine Bordes. Learning end-to-end goal-oriented dialog. *arXiv:1605.07683*, 2016.

[10] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.