Reinforcement Learning for Multi-Agent Semi-Cooperative Games

Daniel Bodea
danielbodea@college.harvard.edu

Aditya Dhar
adhar@college.harvard.edu

Bill Zhang
billzhang@college.harvard.edu

David Zhu
dzhu1@college.harvard.edu

Abstract

The successful application of reinforcement learning principles to adversarial and cooperative games, like Poker, Chess, and various Atari games suggests an opportunity to extend reinforcement learning to the area of "semi-cooperative" games, where agents are placed in long-term competition that requires some level of collusion within stages of the game. We identify one such "semi-cooperative" game, Finding Friends, and establish a series of mechanisms to approximate the environment of the game in order to engender semi-cooperative behavior. We then introduce generative and DQN models for reinforcement learning agents to play the game under these mechanism, and analyze the insights they provide into the complex dynamics of this setting. Finally, we lay the groundwork for future work to extend and improve our results.

1 Introduction

Reinforcement learning literature primarily focuses on mixed cooperative-competitive environments by identifying agents as either cooperative or adversarial. We add to this base by identifying "semi-cooperative" games where immediate cooperation is necessary, but long-term selfish behavior is also crucial for success. We identified Finding Friends (FF), a version of Tractor, as the necessary testing ground for semi-cooperative games, as there are several players on dynamic teams, each ultimately attempting to raise their individual levels, but cooperation with the team in each round is necessary to raise each individual's level.

Our goal is to apply reinforcement learning to create agents that perform well in this semi-cooperative setting, first building a simple model, then extending and generalizing it with different mechanisms including team cooperation and selection to ultimately resemble the complete game of FF.

1.1 Background

We denote the basic environment of FF as having 5-12 players, two decks of cards, two teams (the "kingship", selected by the "king" every round, and the "peasantry", which is everyone not in the kingship), and ending when one player reaches the highest possible level, known as the level cap (ordinarily 13). Every round, a king is selected, who then is allowed to pick a number of teammates (in the -player version, one teammate) to join their team.

In the real-life FF game, the two teams then play out a "round" involving playing cards (one to four of a kind) to beat groups of cards played by other players, the exact details of which we do not go into here and are not relevant, which we can model as a random Bernoulli event of either the kingship winning the round or the peasantry winning the round, depending on the skill levels of the players (and with the randomness arising from the randomness in the dealing of the cards). If the kingship wins the round, then all players on the kingship team level up by one level, while if the peasantry wins the round, nobody levels up. The role of being king is then passed on to the next player in order and a new round begins, until the entire game ends when somebody reaches the predetermined level cap.

1.2 Semi-Cooperative Setting

The semi-cooperative nature of this setting is realized as follows. Generally, when a player is chosen by the king to be on the kingship, the player will cooperate with the king and play well in order to maximize the chance of leveling up. In this case, we have an immediate need to cooperate. However, if the king is only one level away from reaching the level cap, a player on the kingship may instead wish to defect and try to prevent the king from winning it all. This arises from the long-term reward of winning the entire game. Thus, we result in a semi-cooperative setting, where we have short-term incentives to cooperate but long-term incentives to act selfishly.

2 Related Work

Multi-agent reinforcement learning has been studied across many cooperative and adversarial games. In 1995 [8], the Iterated Prisoner's Dilemma (IPD) was one of the first settings for examining reinforcement learning agent behavior where cooperation and competition are simultaneously encouraged [10]. Recently, we have seen work on Pong [9] that used deep reinforcement learning algorithms for different degrees of cooperation and competition, which has further expanded to the study of cooperative navigation, predator-prey relationships, and covert communication.

Tractor has often been studied as an incomplete information game [4]; however, these works study Tractor as a four-player game with very defined incentives to cooperate or compete (cooperate with your team, don't cooperate with other team). FF, as a derivative of Tractor, has an opaque set of incentives that add interesting effects and make it an interesting point of study as a semi-cooperative game rather than an incomplete information game. Since the boundaries between the two behaviors are not well-defined, we aim to document the behavior of the reinforcement learning agents in terms of their ability to balance the tension between working together while not letting any one player get too far ahead. We identify that this tension similarly exists in Settlers of Catan [3]; however, there is at present no analysis of this semi-cooperative dynamic or how it affects agents' behaviors [7].

3 Approach

We start by implementing the various mechanisms and agents for our iterative approximations of FF, which we outline below. Where appropriate, we provide theorems regarding performance and optimality. We are mainly interested in the way agents behave in response to the mechanisms they face.

3.1 Mechanisms

Recall our objective of creating simpler semi-cooperative universes that gradually approach the full complexity of FF. In particular, for n players, we define a **mechanism** as a randomized algorithm M which takes as input the players that belong to the kingship, namely the king k and the m selected friends $f_1, \dots, f_m$, and outputs the resulting changes in level for all players. In particular, a mechanism must satisfy two invariants: (i) only players on the kingship can increase in level, and (ii) all players on the kingship always increase by the same number of levels.

Here, the first condition is necessary for introducing the cooperative component: being on the kingship together gives one the opportunity to advance towards the ultimate objective. The second condition is necessary for introducing the competitive component: the other players on the kingship will always advance forward and thus increase their own chances of winning. We now introduce the various mechanisms we constructed:

- **Base (Bernoulli) Mechanism**: The kingship advancing in level is modeled as a random variable $X \sim \text{Bern}(p)$, where $0 < p \leq 1$ is a constant. This means the kingship only increases one level at a time. In this basic environment, the choice of friend does not affect the chance of increasing in level.

- **Skill Mechanism**: Each player is assigned a relative skill level $s_i \in [0, 1]$ such that $s_i > 0$ and $\sum_{i=1}^{n} s_i = 1$, and the kingship advancing in level is modeled as a random variable $X \sim \text{Bern}(p(k, f_1, \dots, f_m))$, where p is modeled as a function:

$$p(k, f_1, \dots, f_m) = s_k + \sum_{i=1}^{m} s_{f_i}$$

  Here, the probability of advancing is additive in the skill levels of the members of the kingship. Thus, agents with higher skill level are more advantageous to add to the kingship. More complex models of the function p can be adopted and studied later.

- **Sabotage Mechanism**: Each player additionally toggles a cooperation indicator $c_i \in \{0, 1\}$, such that $c_i = 1$ means they intend contribute their skill level to their respective team (kingship or peasantry), and $c_i = 0$ means they intend to sabotage their respective team by not contributing their skill level. Thus, we now model p as:

$$p(k, f_1, \dots, f_m) = \frac{c_k s_k + \sum_{i=1}^{m} c_{f_i} s_{f_i}}{\sum_{j=1}^{n} c_j s_j}$$

  Here, a member of the kingship's decision to sabotage strictly decreases the probability that the kingship advances in level, while a member of the peasantry's decision to sabotage strictly increases the probability that the kingship advances in level. Thus, agents who are likely to sabotage are disadvantageous to add to the kingship.

- **Poisson Mechanism**: Each player on the kingship now has the possibility of leveling up more than once, meaning the kingship advancing in level is now modeled as $X \sim \text{Pois}(p(k, f_1, \dots, f_m))$, where p can be modeled in any of the above ways. This increases the variability of leveling up to more closely reflect that of the actual game.

3.2 Baseline Agents

In order to assess the viability of our reinforcement learning agents, we construct several baseline agents that provide intuitive behaviors for each of the above mechanisms. Each agent is currently only responsible for choosing friends when they are king, as well as deciding whether to sabotage given that they are selected as a friend of the king. All baseline agents never choose to sabotage.

- **Basic Agent**: The agent selects friends at random from among the other players, thus each player is chosen as a friend with probability $\frac{m}{n-1}$. This agent represents zero sophistication.
- **Lowest Level Agent**: The agent selects as friends the other players who currently have the lowest levels. We conjecture that this agent is optimal under the Base Mechanism, as there is no distinction between players besides their respective levels, and thus the best players to choose are those who are farthest from winning.
- **Strategic Skilled Agent**: The agent selects as friends the other players with the highest skills that are currently at a level lower by a set difference than itself. We cannot confirm the optimality of this agent under the Skilled mechanism, but conjecture that the optimal strategy when skills are present is to maximize the probability of winning a round by picking the highest skilled agent while buffering against more skilled agents overtaking in levels and winning the game. This is especially important where level cap is sufficiently low; higher-skilled agents achieving points in a team increases the probability that said agent wins the game. As a result, the strategic skilled agent picks agents in order of decreasing skill after ensuring that a difference in levels is met.

3.3 Generative Agents

The Strategic Skilled Agent above is constructed with knowledge of the skill levels of the other players. However, it may be that those skill levels are not known, but rather must be learned throughout the duration of the game. We thus introduce a generative model that an agent can use for learning the skill levels of other players as the game goes on. For the sake of simplicity, we use $m = 1$ for our below analysis:

- **Beta-Binomial Agent**: Under any of the Bernoulli Mechanisms, the agent would like to learn the unknown probability $p(k, f_1)$ of leveling up having chosen a given friend f_1. If the agent can learn this function, the agent can then solve for s_{f_1} since it knows its own skill level s_k. We first model the prior distribution as $p(k, f_1) \sim \text{Beta}(\alpha, \beta)$ for some choice of initial parameters. Then, for each instance t where the agent chooses f_1, let r.v. X_t represent the observation of whether the kingship leveled up or not. The posterior distribution can thus be updated with:

$$p(k, f_1)|X_1, \ldots, X_T \sim \text{Beta}\left(\alpha + \sum_{t=1}^{T} X_t, \beta + T - \sum_{t=1}^{T} X_t\right)$$

  As proven for the Beta-Bernoulli conjugacy. As the number of trials $T \to \infty$, $p(k, f_1)|Y_1, \ldots, Y_T$ converges quickly to the true probability, and thus the skill level of the given friend can be learned.

- **Gamma-Poisson Agent**: Under any of the Poisson Mechanisms, an identical procedure can be used for the agent to learn the probability $p(k, f_1)$ of leveling up having chosen the given friend f_1. For a prior distribution $p(k, f_1) \sim \text{Gamma}(\alpha, \beta)$, and for similarly defined X_t representing the observation of the kingship leveling up, the posterior distribution can be updated with:

$$p(k, f_1)|X_1, \ldots, X_T \sim \text{Gamma}\left(\alpha + \sum_{t=1}^{T} X_t, \beta + T\right)$$

  Thus, $p(k, f_1)|Y_1, \ldots, Y_T$ will similarly converge to the true probability as $T \to \infty$, and the skill level of the given friend can be learned.

Based on this generative model, the agent can then use the estimate:

$$\hat{s}_{f_1} = p(k, f_1)|Y_1, \ldots, Y_T - s_k$$

for the relative skill level of each friend f_1, and thus proceed to implement the optimal policy of the Strategic Skilled Agent above.

3.4 DQN Agents

We now turn to the reinforcement learning agents, for which we will apply the Deep Q-Learning Network method. In particular, we start by defining a state as $s = (L, \{\ell_i\}_n)$, where L is the level cap that a player must reach to win, and each ℓ_i is the current level of player i. We again focus our attention on selecting only $m = 1$ friends, and we define an action as $a \in \{1, \ldots, n-1\}$ specifying the player who the king selects as their friend. Note that the king is not allowed to select themselves as the friend. We specify several possible reward schemes below for this framework.

We draw inspiration from [6] in designing a deep Q-learning agent for this task. In particular, we construct a single-layer neural network that takes as input the state s and outputs a probability distribution over the other players, which represents the preferences for choosing them as friends. The player corresponding to the maximum of the probabilities is then chosen as the friend. An ϵ-greedy approach is used to train off-policy, and we start by training a single DQN agent amongst several baseline agents. We similarly implement a memory replay buffer to break the correlations between state-action pairs, and adopt a framework of continuously training a policy agent, while periodically updating a target agent.

3.5 Reward Functions

We implemented four different types of reward/payout functions for our model: Winner-Take-All, Proportional, Hybrid, and Ranked. Each type of reward function has its benefits and drawbacks, which we describe below. In all scenarios, we declare that the game has ended at the end of a round if it is the first time that a player has reached a level that is at least the level cap.

- **Winner-Take-All**: At game end, the winner (or winners, if multiple players reach the level cap at the same time) receives a reward of 1, while the other players receive payouts of 0.

Table 1: The original and estimated skill levels of all players, made by the Beta-Binomial Agent (own skill level is s_5) after training for 1,000 iterations.

	s_1	s_2	s_3	s_4	s_5
Original	0.1	0.3	0.3	0.2	0.1
Estimates	0.138	0.289	0.277	0.196	0.1
Original	0.4	0.1	0.1	0.2	0.2
Estimates	0.356	0.131	0.129	0.184	0.2
Original	0.125	0.250	0.375	0.125	0.125
Estimates	0.158	0.232	0.343	0.142	0.125

This reward function most accurately portrays the real-life objective of playing FF, where the ultimate goal is to win the entire game (and there is no reward for placing second). However, because the reward is only realized at the end of the entire game, learning can be slow and have high variance. Furthermore, a strategy that does well but gets unlucky at the end and places second receives no positive feedback, which is also problematic.

- **Proportional**: At game end, each player i receives a reward proportional to their level:

$$R_i = \frac{\ell_i}{\sum_j \ell_j}$$

This reward function provides direct feedback to the model as to how well it has done relative to the other players. However, it fails to accurately capture the overall objective of the game, which is to be the first to reach the level cap.

- **Hybrid**: At the end of each round, any player who leveled up that round receives a reward of 1. At the end of the entire game, each of the losers receives a payout of negative level cap. This reward function acts as a bit of a hybrid between Winner-Take-All and Proportional, in that losing a game results in a large negative payout, while the rest of the reward is distributed proportionally to the number of levels gained.
- **Ranked**: At game end, each player receives a reward equal to the total number of players minus the number of players at a higher level. This reward function essentially "ranks" the players (without regard to their exact levels) and assigns rewards based on those rankings. To provide an extra incentive to win the entire game, we can skew the rewards exponentially by taking 2 (or some other constant) to the power of this original Ranked payout, so instead of payouts of 5, 4, 3, 2, 1, we would have payouts of 32, 16, 8, 4, 2. By ranking the players, this reward function removes small variations in payout based on slightly different final levels that don't actually matter, while still providing sufficient feedback and capturing the importance of winning the entire game.

We ran experiments under all four reward functions and compared how well the agents learned under each setting. We expected the Hybrid and Ranked reward functions to work best since they provide meaningful feedback to the agent while still encouraging it towards the original objective of winning the entire game.

4 Experimental Results [1]

4.1 Generative Agents

We performed a series of experiments using the Beta-Binomial Agent under a Bernoulli Skill Mechanism. This environment is designed so the players all have different skill levels, and thus the Beta-Binomial Agent must form accurate estimates of their skill levels in order to well. Running for 1,000 iterations under a variety of skill level distributions, we see that the Beta-Binomial Agent successfully approximates the skill levels of the other players (see Table 1). The Beta-Binomial Agent was able to achieve this accuracy regardless of the sophistication of the other agents. Similarly accurate approximations were observed for the Gamma-Poisson Agent under the Poisson Skill Mechanism.

[1] All code for this project can be found at `https://github.com/bzhang42/Finding_Friends`

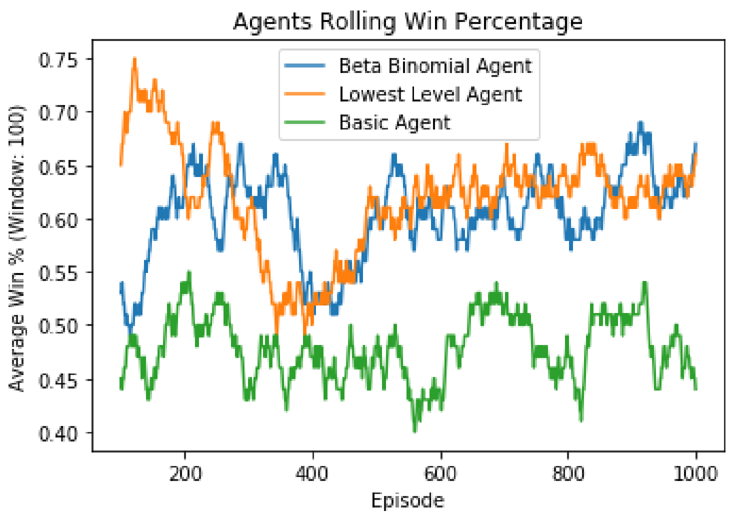


Figure 1: Win percentage of various agents under a Bernoulli Skill Mechanism with $n = 5$, $p = 0.4$, and $L = 10$, and skill levels $\{1/15, 2/15, 3/15, 4/15, 5/15\}$ with a rolling window of length 100.

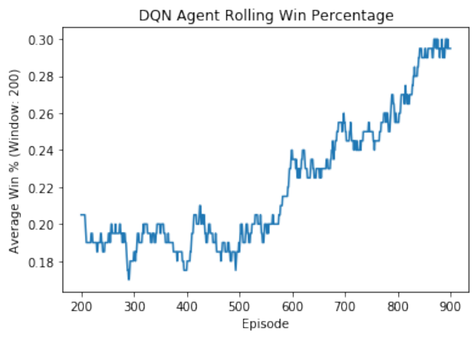
We then proceeded to compare the generative agents' performance to that of the Basic and Lowest Level Agents under a Skill Mechanism. For all of these experiments, we used a control group of four Basic Agents and a fifth selected from the set {Beta-Binomial, Lowest Level, Basic}. We then set the skill levels of the agents to $\{1/15, 2/15, 3/15, 4/15, 5/15\}$, so that the agent in question would have a non-trivial advantage to be gained from accurately estimating the other players' skill levels. The results are depicted in Figure 1. Unsurprisingly, both the Beta-Binomial and Lowest Level Agents outperformed the Basic Agent significantly. Surprising, however, was the fact that there was no distinguishable difference in performance between the Beta-Binomial and Lowest Level Agents. We conjecture that, under the Skill Mechanism, the advantage gained from accurately identifying agents with high skill levels is offset by the fact that those agents are most likely ahead in the game – thus, selecting agents which are currently at lower levels is more critical, leading to mitigated benefits from the accurate estimation.

4.2 Reinforcement Learning Agents

We performed experiments with the DQN Agent in the same manner as above: with four Basic Agents under a Base Mechanism, $p = 0.4$ and $L = 10$.

We trained and tested our model with three algorithms: RMSprop, SGD with momentum, and Adam. We use RMSprop because of its focus on adaptive learning rate methods, SGD because of its ability to iteratively train our model, and Adam because the advantages of RMSprop and SGD with momentum. We tuned SGD's hyperparameters including learning rate and momentum. When testing, we examined learning rates of 0.1, 0.01, and 0.001 and momentum of 0.9. We found that learning rates did not have pronounced effects on the rolling win percentage. With Adam's learning rate hyperparameter, we also tested learning rates of 0.1, 0.01, and 0.001 and found no pronounced effects. During our experiments, we kept track of the rolling win percentages, which are most indicative of the agent's performance. We originally considered tracking average score at the end of each game; however, we observed that oftentimes, the suboptimal Basic Agents would have higher average scores but lower win percentages than the more optimal Agents, so we discarded this metric as not robust.

We observed the DQN agent successfully learn in several configurations (see Figure 2), but definitely not across the board. In an effort to bound state size, we performed additional experiments while reducing the number of players to $n = 3$. This would have reduced our state size to $1,000$ states. In an effort to reduce the impact of noise from the other agents' actions on the DQN Agent's learning, we also restructured the DQN algorithm so that, despite the immediate next state not being actionable, the DQN Agent is still able to observe and save the state to its replay buffer. This effectively decreases the noise observed by the agent in transitioning from one state to the next. We also modified the



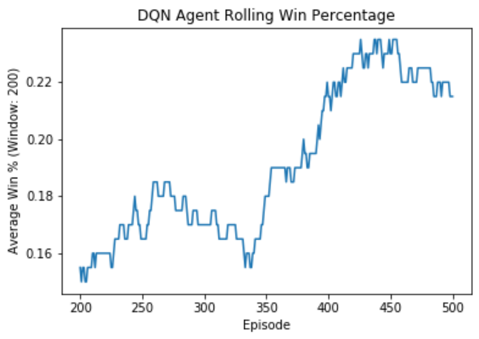


Figure 2: The win percentage of the DQN Agent trained under a Bernoulli Base Mechanism with $n = 5$, $p = 0.4$, and $L = 10$, under RMSProp (left) and SGD with momentum = 0.9 and $\alpha = 0.1$ (right), with a rolling window of length 200. For reference, the Lowest Level Agent achieves a win percentage of 0.30.

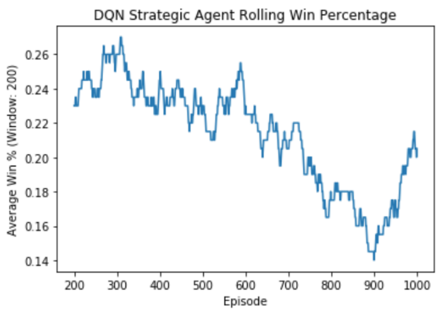
reward function using the above methods in an attempt to better reflect the incentives of the game. Through all these instances, we found that performance between three and five players did not vary significantly, despite lengthening the number of iterations from 500 to upwards of $2,000$. We discuss reasons for these limitations in performance below.

We finally constructed a modified DQN agent for the Skill Mechanism which takes advantage of the logic behind the Strategic Skilled Agent. The specific implementation of the mechanism required altering the standard DQN Agent, which usually maximizes expected reward by learning from past actions. We modify it such that the agent follows the Strategic Skilled Agent rather than picking the maximal expected reward option when the two were not the same. The challenge faced, however, is that any deviation from the standard DQN requires explicitly not maximizing expected reward at each step. This risks the agent learning incorrectly to not pursue optimal policy. Figure 3 suggests that the agent 'learning' is counterproductive and decreases the win rate, although the rolling-win average in the end of the graph increases significantly; the win rate percentage over the last hundred episodes averages 30% in this graph, which is a higher win percentage than the previous 200-length windows plotted. To limit potential bad learning, we restrict the levels at which the agent uses the strategic mechanism. After experimentation, we note that the Strategic Skilled Agent's policy should not be employed very early or very late in the game, when the agent requires maximizing expected return to out-compete skilled agents to gain an early lead or win the game.

5 Discussion

One of the challenges we faced in our experiments was the large amount of noise our agent encountered during training. Our agent is only able to perform an action when it's their turn to be king and select friends. Thus, after our agent performs an action, its next actionable state comes after all of the other agents have performed an action, and so the next state our agent encounters reflects the collective noise of the other agents' (often random) actions. It is therefore difficult for our agent to learn the direct impact of their action amongst the distractions that the other agents impose. To fix this, we modified the DQN algorithm to pass the immediate next state, rather than the next actionable state, to the agent each time it took an action. That way, there would be a much clearer connection between the agent's action and its impact on the state of the game.

Another challenge we faced was the large number of state spaces we could encounter. If n is the number of players and L is the level cap, then we could potentially encounter up to L^n unique state spaces in the course of training. Throughout most of our trials, we kept $n = 5$ and $L = 10$, resulting in 100,000 state spaces. This large number of state spaces could make it difficult for our agent to reliably learn the correct policy in each of the states. However, reducing the number of states by lowering L makes the game less realistic and more prone to random chance, since it is much easier for a player to get lucky and hit the level cap when the level cap is low. Thus, we encounter a tradeoff between learnability and variance.



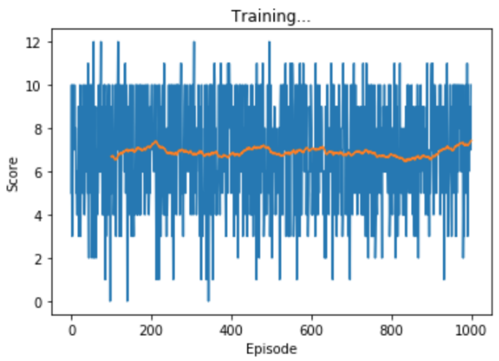


Figure 3: The DQN Agent under a Poisson Skill Mechanism with $n = 5$, $p = 0.4$, and $L = 10$, using RMSProp, with win percentage over a rolling window of length 200 (left) and average final score over a rolling window of length 100 (right) depicted.

Meanwhile, we did observe the agent learning in several instances over a few hundred episodes, despite the state space being so enormous. We conjecture that this is because the vast majority of the states are rarely, if ever, visited. The vast majority of the time, the players' levels are densely centered, resulting in a few states receiving frequent visits while most lay untouched. This makes it plausible that the DQN Agent could learn the correct configuration based off its experience with those more frequented states, and since the optimal relationship here is rather simple (pick the friend with the lowest level), it is possible for the DQN Agent to fortuitously visit the right states and learn the correct relationship.

6 Conclusion

In order to make this work worth sharing with other ML experts, we present a few next steps. First, due to limitations in computing power, we were unable to train for a more robust number of iterations ($\sim 10,000$), but conjecture that the DQN Agent would likely learn better given a longer horizon, since it would smooth out the noise and randomness inherent in the game. Second, we did not analyze the performance of any sabotage agents, as the inconsistent performance under the Base and Skill Mechanisms required our attention more urgently. The state space for the Sabotage Mechanism would be larger, as it would require specifying the king of any given round, thus increasing the difficulty of learning significantly. However, it goes without question that analyzing sabotage behavior would lead to even more interesting discussions on semi-cooperative tendencies in reinforcement learning agents. We would also like to experiment with greater variation in implementation of the skilled mechanism to verify theorized optimality in the game both with and without learning.

Ultimately, this project has paved the initial groundwork for an exploration of the semi-cooperative space of Finding Friends. Our analysis of generative and DQN agents under the Skill Mechanism provides an initial intuition into the trade-offs between cooperating in the short term to level up together, and competing in the long term to reach the level cap first. In addition, our construction of baseline mechanisms and agents provides an important learning curve for others hoping to approximate the behavior of agents in simpler environments while gradually increasing the complexity and resemblence to Finding Friends. We are excited to continue exploring this game and the implications of reinforcement learning agents on this field of study.

References

[1] H. Charlesworth. Application of self-play reinforcement learning to a four-player game of imperfect information. *CoRR*, abs/1808.10442, 2018.

[2] P. J. Hoen, K. Tuyls, L. Panait, S. Luke, and H. L. Poutré. An overview of cooperative and competitive multiagent learning. In *LAMAS*, 2005.

[3] S. Hoover. New sheng ji gaming bot will advance machine learning. 2019.

[4] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[5] Q. Ma and H. Nash. Solving multiplayer games with reinforcement learning. 12 2008.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[7] M. Pfeiffer. Reinforcement learning of strategies for settlers of catan. In *International Conference on Computer Games: Artificial Intelligence, Design and Education 2004*, pages 384–388. Univ. of Wolverhampton, 2004.

[8] T. Sandholm and R. H. Crites. On multiagent q-learning in a semi-competitive domain. In *Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*, IJCAI '95, pages 191–205, Berlin, Heidelberg, 1996. Springer-Verlag.

[9] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *CoRR*, abs/1511.08779, 2015.

[10] L. Xue, C. Sun, D. Wunsch, Y. Zhou, and F. Yu. An adaptive strategy via reinforcement learning for the prisoner's dilemma game. *IEEE/CAA Journal of Automatica Sinica*, PP:1–10, 03 2017.

Appendix

All code for this project can be found at `https://github.com/bzhang42/Finding_Friends`.