Everyone Follow the Rules of the Road

Gabe Smithline

May 2024

1 Introduction

Over the years, interest in AI has grown exponentially. It is a highly interdisciplinary field based in computer science but expanding into numerous other domains. As AI has grown, many fields such as neuroscience, mathematics, philosophy, and economics have influenced its development. In reinforcement learning, specifically multi-agent reinforcement learning (known as MARL), there has been a lot of very successful research regarding using ideas from mathematics and economic theory; mainly ideas from game theory and mechanism design. The use of game theory in other parts of AI and computer science in general has been steadily growing (it even has its own name for a subfield, EconCS). In the reinforcement learning space, this intersection has been called Empirical Game Theoretic Analysis (EGTA), it has led to many interesting findings and is one of the major areas of focus for DeepMind, OpenAI, and Anthropic as each has multiple game theory groups. My goal here is to explore this field of EGTA, specifically one tool that has emerged from it, Policy Space Response Oracles, and use it in an environment in which it has not been used before, mainly in a normative Markov game environment.

2 Background

2.1 The Environment and Normative MDP

For this project, I will be making use of the MA-Gym intersection environment [Koul(2019)], as an environment. Driving, specifically an intersection, seemed like a natural choice as there are specific rules one follows, but there are also specific norms one follows to be a better driver. In this environment, there are 10 or 4 cars, each trying to get to its destination at the intersection. We can more formally express the Normative Markov Game as the following:

- **Agents (A):** The 10 or 4 cars operating in the environment, with a maximum of n_{max} cars being active at any point. Each car is represented by a one-hot binary vector set $\{n, l, r\}$ encodes a unique ID, current location tuple, and assigned route number respectively. Each agent controlling a car can only observe other cars in its vision range (the surround 3 by 3 neighborhood) [Koul(2019)].
- **Actions (A):** The action space consists of "gas" (move forward) and "brake" (stay in place). Actions are taken based on the current state and the policy the agent follows, with influence by norms.

- **Reward Function (R):** $R(s, a, s")$ defines the immediate reward received after transitioning from state s to s' by taking action a. In my setup, this includes rewards or penalties associated with reaching destinations, avoiding or causing collisions, adhering to or violating traffic norms (e.g., yielding, not braking unnecessarily, efficiently crossing the junction), and additional rewards for norm compliance. For example, if a collision occurs between two cars then a reward is given of -10 but does not affect the simulation. To discourage traffic jams each car gets a reward of $\tau \cdot r_{time} = -0.01\tau$ at every time step, where τ is the number time steps passed since the car arrived, so we can express total reward at time t as:

$$r(t) = C^t \cdot r_{car} + \sum_{i=1}^{N^t} \tau_i \cdot r_{time} + r_{Norm}(s_t, a_t)$$

  where C^t is the number of collisions occurring at time t and N^t is the number of cars present [Koul(2019)].

- **Transition Function (T):** $T(s, a, s") = P(s"|s, a)$ is the probability of transitioning from state s to state s" given action a. In a deterministic setting like my grid environment, this can be simplified to direct state transitions based on actions and the layout of the environment, including entry and exit points and possible collision/norm outcomes.

- **Norms (N):** These are a set of obligations and prohibitions on all agents, which affect their reward structure, which then influence the transition function indirectly. For instance, obligations to yield at certain points or prohibitions against driving a certain way can lead to different state transitions and rewards. Norm violations can incur penalties, thereby influencing agents' decision-making processes.

- **Policies** (π)**:** A policy $\pi(a|s)$ defines the strategy that an agent follows, determining the probability of taking action an in state s. The policy could for example be meant to maximize long-term rewards while considering compliance with norms.

- **Discount factor** (Υ)**:** Simply a factor indicating the importance of future rewards.

- **States (S):** The state space in the environment is defined by the positions of all cars on the grid, their directions, and their intended routes. Each state also encapsulates norm compliance. The state vector s_j for each agent is then a concatenation of all these vectors, having dimensions $(3^2) \times (|n| + |l| + |r|)$ [Koul(2019)].

2.2 Norm Structure:

Norms can be more formally described as a series of temporal logic statements that are evaluated at each step. Simply temporal logic statements allows for the construction of specific norms in the system and customization of norms in the system. These logical statements can be described as the following:

Temporal Logic Norm (LTL)	Description
$\Box(\mathit{collision} \rightarrow \Diamond \mathit{penalty})$	Always, if a collision occurs, then eventually a penalty is applied.
$\Box(\mathit{not\_on\_track} \rightarrow \Diamond \mathit{return\_to\_track})$	Always, if an agent is not on the track, then it must eventually return to the track.
$\Box((\mathit{yield\_right} \wedge \Diamond \mathit{approaching\_intersection}) \rightarrow (\neg \mathit{move\_forward} \cup \mathit{path\_clear}))$	Always, if an agent needs to yield right and is approaching an intersection, it should not move forward until the path is clear.
$\Box(\mathit{approaching\_turn} \rightarrow \mathcal{X}(\mathit{slow\_down} \wedge \mathcal{X}(\mathit{turning} \wedge \mathcal{X}\mathit{accelerate})))$	Always, if approaching a turn, then in the next step slow down, and after turning, accelerate in the subsequent step.
$\Box(\mathit{path\_clear} \rightarrow \mathcal{X}\mathit{accelerate})$	Always, if the path ahead is clear, then accelerate in the next step.
$\Box(\mathit{at\_destination} \rightarrow \Diamond \mathit{exit})$	Always, if at destination, then eventually exit.
$\Box(\mathit{off\_track} \rightarrow \Diamond \mathit{on\_track})$	Always, if off track, then eventually must return on track.

Table 1: Temporal Logic Norms for Traffic Junction Scenario

3 Policy Space Response Oracle Algorithm

Policy Space Response Oracles (PSRO) [Lanctot et al.(2017)Lanctot, Zambaldi, Gruslys, Lazaridou, Tuyls, Pérolat, Silver, and Graepel] builds on double oracle algorithms and population based algorithms that can learn policies in general sum games with full or partial observability of two or more agents, it builds on Empirical Game Theoretic Analysis (EGTA) [Albrecht et al.(2024)Albrecht, Christianos, and Schäfer]. In this algorithm we construct what's called a meta-game, as an abstraction of the underlying game (it's essentially a finite normal form game where each action corresponds to a specific policy π_i in the underlying game), and then we essentially apply some equilibrium analysis/game solver to the metagame. The reward function in the meta game $R_i(\pi_1, ..., \pi_n)$ gives the average return for the agent i in the game if the agent decided to select polices $\pi_1, ..., \pi_n$ [Albrecht et al.(2024)Albrecht, Christianos, and Schäfer].

PSRO constructs these meta-games in each generation of population training to evaluate and grow the possible policy populations [Albrecht et al.(2024)Albrecht, Christianos, and Schäfer]. The algorithm can be expressed below:

Algorithm 1 Policy Space Response Oracles (PSRO) for Traffic Junction with LTL Norms

```
1: Initialize policy populations $\Pi_i^1$ for all agents $i$ with random or heuristic-based policies.
2: **for** each generation $k = 1, 2, 3, \ldots$ **do**
3:     Construct meta-game $M^k$ from current populations $\{\Pi_i^k\}_{i \in I}$.
4:     Use meta-solver on $M^k$ to obtain policy distributions $\{\sigma_i^k\}_{i \in I}$.
5:     **for** each agent $i \in I$ **do**
6:         **Train best-response policies considering LTL norms:**
7:         **for** each episode $e = 1, 2, 3, \ldots$ **do**
8:             Sample policies for other agents $\pi_{-i} \sim \sigma_{-i}^k$.
9:             Use PPO $\pi_i^{"}$ w.r.t. $\pi_{-i}$ in the underlying game $G$.
10:        **end for**
11:        Grow population $\Pi_i^{k+1} \leftarrow \Pi_i^k \cup \{\pi_i^{"}\}$.
12:    **end for**
13: **end for**
```

3.1 Expression of the Intersection:

In the algorithm, we simply express our meta-game as a tensor containing multiple bimatrix games. To build this tensor, we sample the environment to construct the meta-game between each agent permutation of size 2. This creates a simple model of the intersection. Here is a simple example:

Bimatrix Game:

	Go (Car 2)	**Stop (Car 2)**
Go (Car 1)	$(-10, -10)$	$(5, 0)$
Stop (Car 1)	$(0, 5)$	$(1, 1)$

You'll notice in this game, there is no dominant strategy. To solve it and compute a Nash equilibrium policy, we employ Lemke Howson's algorithm of Best Response Polytopes. This guarantees to find a Nash equilibrium if the game is not degenerate. The algorithm begins with finding an equilibrium where each player associates a normalized weight across actions they can play, then it employs pivot rules similar to linear programming, where it involves moving from one vertex of the strategy space to an adjacent one. Each strategy and their corresponding payoff is initially labeled, these labels represent if the strategy was played. Now, the key part of the algorithm is the path following, where it follows the strategy polytope (which is just a geometric representation of strategy mixes), it does this by alternately switching strategies for each player in way that maintains feasibility and seeks to satisfy the conditions of a Nash equilibrium. It terminates when it reaches a vertex of the polytope and all labels cannot be reassigned as each player would be worse off.

3.2 Proximal Policy Optimization (PPO):

After we've constructed our meta-game policies, we then train our proximal policy agents. In PPO, at a high level, it consists of an actor (a policy network) and a critic (a value network). In the PSRO the actor takes the Nash equilibrium policies and truncated state information as input, to which then it produces a distribution over actions. The critic takes state-action pairs, then produces a q-value for each action in the state. In the PPO algorithm we also story history, here this consists of: the action, Nash policy, if the agent completed its trip, its position and the direction it's headed. From this we pass it to the critic where we compute our advantage, which is expressed as:

$$A_t = R_t + \gamma V(s_t + 1) - V(s_t)$$

the advantage calculates the critic's value estimates, it is the rewards at time step t plus the discounted difference in the estimated values at time $t+1$ and time t in state s; it's measuring the quality of the action taken compared to the average action at that state as estimated by the critic. The actors' policy is then updated using a gradient ascent method, where the gradient is being scaled by the advantage. This update is aiming to increase the probability of actions that lead to higher than expected returns and decrease the probability of lower than expected returns. What's also key is the loss function for the attacker, it can be expressed as:

$$L^{\text{Clip}}(\Theta) = \hat{\mathbb{E}_t} = \left[\min\left(r_t(\Theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right]$$

$$r_t(\Theta) = \frac{\pi_\Theta(a|s)}{\pi_{\text{old}}(a|s)}$$

Here $\pi_\Theta(a|s)$ is the probability of taking action a in state s under the new policy parameterized by the weight vector Θ, while $\pi_{\text{old}}(a|s)$ is the probability under the old policy parameters Θ_{old}. To prevent the policy from changing too much, PPO makes use of clipping where $\epsilon = .2$ usually, this prevents the ratio from deviation too far from 1, essentially if the ratio leads to a large value increase in the policy the clipped value will be used to counteract that and over updating the network.

4 Results

Experiments were run with 300 generations, and 100 episodes for each generation.

4.1 4 Agent Experiment:

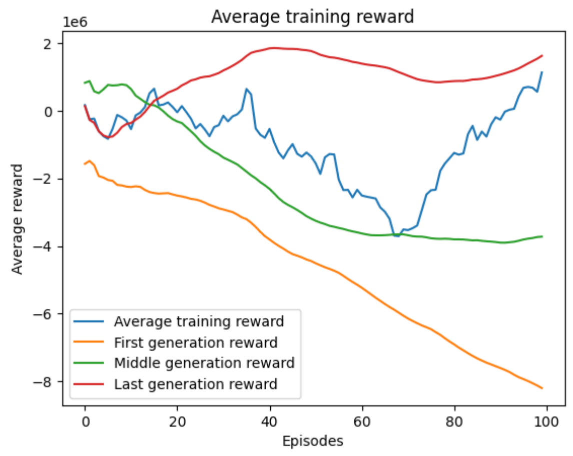

Figure 1: Rewards 4 Agents.

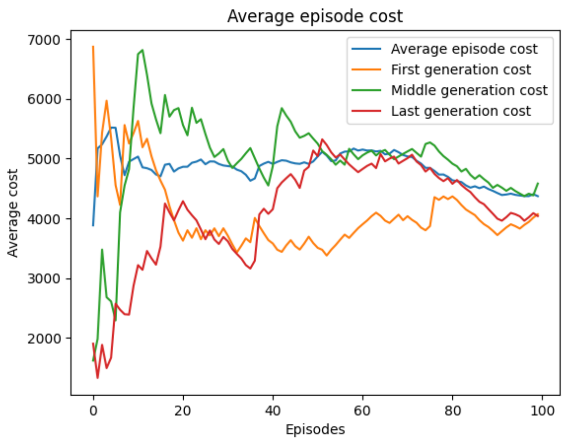

Figure 2: Norm Costs 4 Agents.

Here we see the results for our 4 agent training. We see on the right the average reward per agent over each episode, we have 4 different lines. The first generation, the 150th generation, the 300th generation, and the average across all generations. We see that as generations continue, our rewards accumulated from the environment increase and our agents are performing better. This is a strong indication that our algorithm seems to be somewhat successful, as the rewards increase as the agent generation increases. Even in this multi-agent stochastic environment, we're able to reasonably find strategies to navigate it. On the right we see a similar chart, this is the average cost of norm violations over the course of the generations, we again see the 4 same lines. What's interesting here is all seem to converge around certain the same violation costs, there is a multitude of reasons for this, which we will discuss below.

4.1.1 4 Agent Simulation Runs:

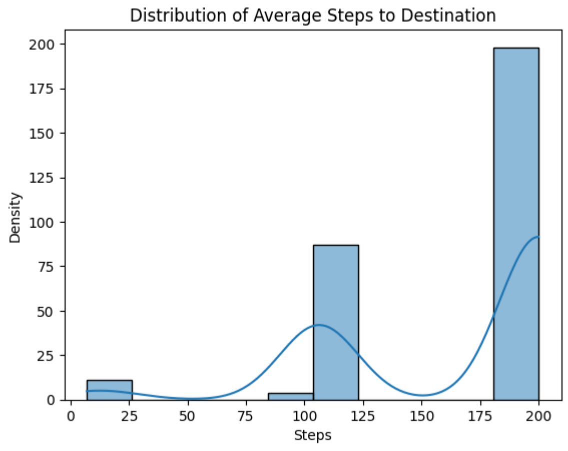

Figure 3: Number of Steps to Destination, 4 Agents.

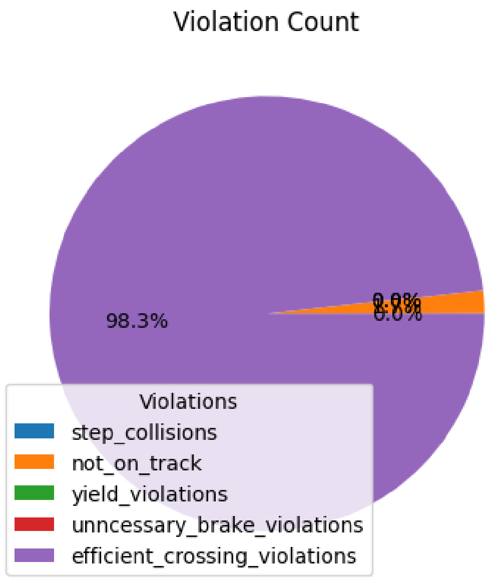

Figure 4: Breakdown of Norm Costs in Simulation 4 Agents.

4.2 10 Agent Environment:

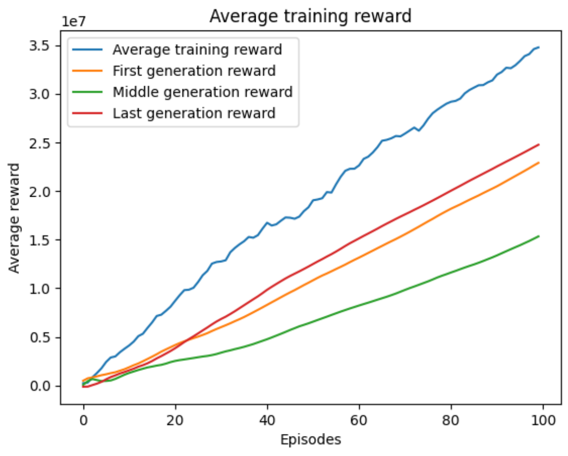

Figure 5: Rewards 10 Agents.

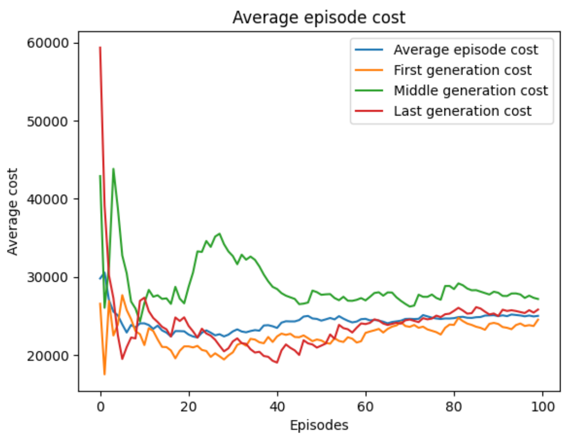

Figure 6: Norm Costs 10 Agents.

Again on the left we see the same graph but now for 10 agents. We see here that the average trend is above the final generation. This indicates that there were generations before the final one that actually performed better. This possibly could be a result from over training the agents, so as they do not generalize as well, also it could just be a result of the environment, or a combination of the two. This environment is a stochastic multi-agent environment, not only does each agent enter at a random location and need to navigate, each is given a random destination to find, meaning at each episode each agent is having to make an entirely different set of decisions to accomplish its goal. It needs to navigate from a new location, and it needs to navigate through the 9 other agents who are doing the same. But what's reassuring is that all generations seem to be linear, meaning as the episodes go, even though the environments can be drastically different, we're performing better, having less violations, reaching our goals.

4.2.1 10 Agent Simulation Runs:

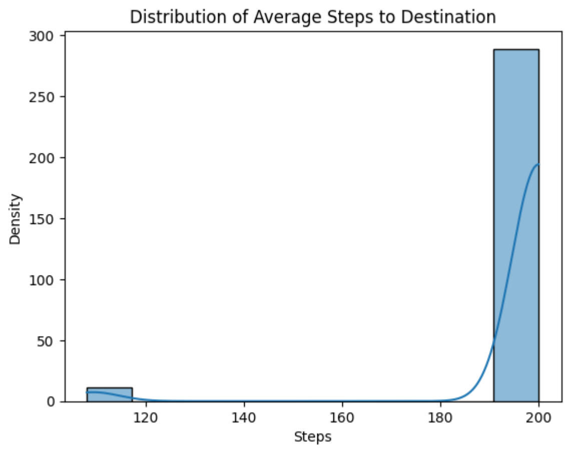

Figure 7: Steps 10 Agents.

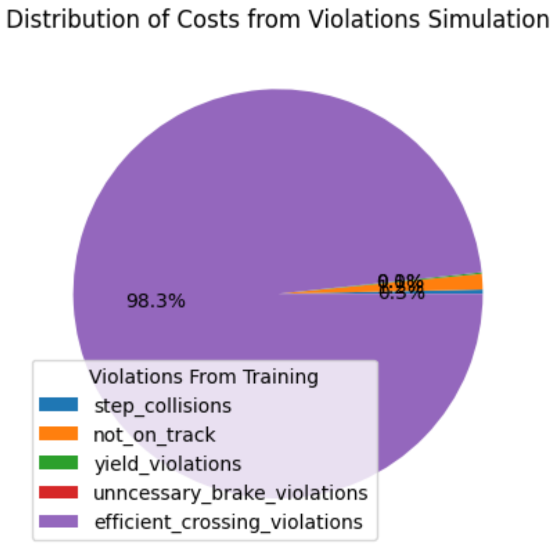

Figure 8: Norm Costs 10 Agents.

Here we see the 10 agent simulation run results with the trained agents. In almost all agents, each simulation run took full 200 steps to reach their destinations, we also see the costs from violations follows a somewhat similar distribution, where the most common violation is efficient crossing violations.

4.3 Comparing Simulation Results in 10 Agent vs. 4 Agent:

We see that the most common violation is efficient crossing violations, one gets a slight penalty for stopping while in the intersection or if their path is open, and they are parked. This norm was implemented with the goal of reducing possible congestion. But we see in both cases it is by far the most common violation. This could be for many reasons, first it could just be that it's actually really not worth it to be more aggressive, the slight penalty is not a significant enough punishment relative to other things that could happen. Also, it might have to do with the meta-game, where in the metagame, the follower has a dominant strategy of stopping, and this is used as input to the proximal policy network. One other thing to note is that there were other violations, but crashing almost never happened in the 4 agent case and was rare in the 10 agent case, which surprised me. Thinking about it deeper, though, it makes sense; if there are not many cars on the road, then many crashes won't happen. In the case of 10 cars, crashes happen, but we punish the network so heavily that each agent quickly learns to be more careful and that maybe being less efficient when crossing is actually better than being more aggressive and risking a crash.

4.4 Transfer Learning Experiment

In this experiment, we trained our agents in a 10 agent environment, took the 4 best agents (determined by reward accumulated), then ran these top 4 trained agents in the 4 agent environment to experiment to see if there were any major deviations compared to the agents in the 4 agent environment. My thought was possibly there would be different results as they were trained in an environment where they had to navigate many other agents. But I was wrong, they did almost entirely the same. This led me to question why, as the learning rate over episodes was different. Thinking about it deeply, it does make more sense. A 4 agent environment has different challenges than a 10 agent environment. In a 10 agent environment you are very concerned about crashes, hence you might experience more, but since we punish the network heavily for crashes, those quickly dissipate. Also, if we have 10 agents in some sense the environment is constrained, there are less spots an agent can explore because there are so many other agents, hence its exploration in some ways can be more efficient if we are able to get over the crashing (that's why we see agents in the 10 agent environment have such a rapid increase of rewards). In 4 agent environments there is just a much larger state to explore as there are fewer cars; agents can get lost, and do take more cirquiticse routes to their destinations.

5 Challenges, Constraints, and Next Steps:

There were a few constraints I had to deal with on this project. The first one being computed resources. I was running my simulations in google collab and on my machine, it took hours to run, which was a bottleneck on the data collected. I also faced issues regarding the single agent learning. First I was using DQN for each agent, it was clear that the agent learned, but that learning did not transfer from episode to episode. I then switched to PPO and had more success, agents started to clearly learn and did not start over at the beginning of a new episode. One other constraint was the actual environment, I could only test configurations of 4 agents and 10 agents. I found this very restricting and in the future I plan to use much more robust environments. I'd also like to explore different game structures, larger and more robust ones, and different equilibrium concepts; possibly also even exploring using more mechanism design in these algorithms to influence different outputs. Finally, time was a major constraint, there are many other things I'd like to do, for example taking a stochastic processes approach to understand norm violations; to me this seems like a random process, and I'd like to take a look at it from a probability theoretic sense to explore this.

6 Conclusion:

In these experiments, we explored the PSRO framework in conjunction with PPO to train agents in a Normative Markov Game. We employed simple ideas from game theory to construct our meta-game, and used PSRO with PPO to solve them. Our results, although relatively small, were optimistic. We were able to learn as episodes continue, but it left me with really more questions than answers. I'm planning to make a much more thorough analysis and continue this work in my free time in the coming months.

References

[Albrecht et al.(2024)Albrecht, Christianos, and Schäfer] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL `https://www.marl-book.com`.

[Bighashdel et al.(2024)Bighashdel, Wang, McAleer, Savani, and Oliehoek] Ariyan Bighashdel, Yongzhao Wang, Stephen McAleer, Rahul Savani, and Frans A. Oliehoek. Policy space response oracles: A survey, 2024.

[Koul(2019)] Anurag Koul. ma-gym: Collection of multi-agent environments based on openai gym. `https://github.com/koulanurag/ma-gym`, 2019.

[Lanctot et al.(2017)Lanctot, Zambaldi, Gruslys, Lazaridou, Tuyls, Pérolat, Silver, and Graepel] Marc Lanctot, Vinícius Flores Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *CoRR*, abs/1711.00832, 2017. URL `http://arxiv.org/abs/1711.00832`.

[Marris et al.(2021)Marris, Muller, Lanctot, Tuyls, and Graepel] Luke Marris, Paul Muller, Marc Lanctot, Karl Tuyls, and Thore Graepel. Multi-agent training beyond zero-sum with correlated equilibrium meta-solvers, 2021.

[Oldenburg and Zhi-Xuan(2024)] Ninell Oldenburg and Tan Zhi-Xuan. Learning and sustaining shared normative systems via bayesian rule induction in markov games, 2024.

[Perolat et al.(2022)Perolat, De Vylder, Hennes, Tarassov, Strub, de Boer, Muller, Connor, Burch, Anthony, McAleer, E
Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, December 2022. ISSN 1095-9203. doi: 10.1126/science.add4679. URL `http://dx.doi.org/10.1126/science.add4679`.

[Wellman et al.(2024)Wellman, Tuyls, and Greenwald] Michael P. Wellman, Karl Tuyls, and Amy Greenwald. Empirical game-theoretic analysis: A survey, 2024.