A APPENDIX

A.1 ERMAS ALGORITHM PSEUDOCODE

Algorithm 1: ERMAS outer loop: `Planner-OPT`.

Result: Robust planner policy π_p.
Input: Initial planner policy π_p and agent policies $\boldsymbol{\pi}$
for ($i = 1, \ldots, n$) {
  Update parameters of agent policies $\boldsymbol{\pi}$: $\boldsymbol{\theta} \leftarrow$ `Agent-Adv-Search`$(\boldsymbol{\theta})$;
  Update planner policy $\pi_p \leftarrow \pi_p + \nabla_{\pi_p} J_p(\pi_p, \boldsymbol{\pi})$;
}

Algorithm 2: ERMAS inner loop: `Agent-Adv-Search`.

Result: Parameters $\boldsymbol{\theta}$ of agent policies.
Input: Planner policy π_p, reward slack ϵ, trust region radius η, Lagrange multiplier $\boldsymbol{\lambda}$, learning rate α_λ, number of warm-up steps k, number of overall training steps n_{train}, and number of inner learning steps n_{train};
Warm up agent policies $\boldsymbol{\theta}$ with k steps of vanilla policy-gradients on $-J_p(\boldsymbol{\theta}) + \sum_{i=1}^{N} \lambda_i J_i(\boldsymbol{\theta})$;
Copy agent parameters into placeholder parameters $\boldsymbol{\theta'} \leftarrow \boldsymbol{\theta}$;
for ($j = 1, \ldots, n_{train}$) {
  Execute $\boldsymbol{\pi_\theta}$ to accumulate batch of experiences B_0 with mean rewards $\boldsymbol{\mu}_0$;
  Apply standard policy gradient: $\boldsymbol{\theta'} \leftarrow \boldsymbol{\theta'} + \nabla_{\boldsymbol{\theta}} \left(-J_p(\boldsymbol{\theta}) + \sum_{i=1}^{N} \lambda_i J_i(\boldsymbol{\theta}) \right)$;
  for ($i = 1, \ldots, N$) {
    Initialize parameters: $\theta_i^* \leftarrow \theta_i$ *# agent i will unilaterally deviate from $\boldsymbol{\pi_\theta}$*;
    for ($k = 1, \ldots, n_{inner}$) {
      Update $\theta_i^* \leftarrow \theta_i^* +$ TRPO $\left(\theta_i^*, \theta_{-i}, \pi_p, \frac{\eta}{n_{\text{inner}}}, B_{k-1} \right)$ to increase $J_i(\theta_i^*, \theta_{-i})$;
      Execute $\pi_{\theta_i^*}, \pi_{\theta_{-i}}$ to accumulate batch of experiences B_k with mean rewards $\boldsymbol{\mu}_k$;
    }
    Approximate meta-learning correction to parameters: $\theta_i' \leftarrow \theta_i' - \lambda_i \theta_i^*$;
    Update multiplier $\lambda_i \leftarrow \lambda_i - \alpha_\lambda \left(\mu_{n_{\text{inner}}, i} - \epsilon_i - \mu_{0,i} \right)$;
  }
  Copy $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta'}$.
}

A.2 ADDITIONAL RELATED WORK

Robust Mechanism Design Mechanism design aims to design reward functions that have desirable equilibrium features, such as stability (Nash equilibria), truthfulness (second-price auctions), and others (Myerson, 2016). Two-level RL can be seen as designing a mechanism (reward function set by a planner) with adaptive agents. As such, in the two-level RL setting ERMAS learns a robust mechanism. Robust mechanism design (Bergemann & Morris, 2005) has studied how mechanisms can be effective even when the environment is imperfectly known. However, it is often very hard to derive analytical solutions for (robust) mechanisms. Instead, Dütting et al. (2019) used deep learning to learn (close to) optimal auctions, while learning optimal taxes by the AI Economist (Zheng et al., 2020) can be seen as adaptive mechanism design using two-level RL. In this sense, ERMAS learns a robust adaptive mechanism.

Behavioral Economics Behavioral economics has studied human reward functions and how human behavior differs from that of idealized agents optimizing simple utility functions (Pesendorfer, 2006). Simon (1976) decomposed bounded rationality into substantive and procedural rationality. Substantively rational agents have incorrectly reward function estimates (e.g., humans might be more risk-averse than a simulated agent) or who require richer games to describe their them. Procedurally rational agents are irrational agents which violate rationality assumptions or do not maximize for their reward function (Simon & Schaeffer, 1990).

A.3 ADDITIONAL ERMAS MOTIVATION

Many behavioral models, e.g., for self-driving cars or economic policymaking, are trained using RL in simulated environments, because real-world experiments are too expensive, infeasible, or unsafe. However, many RL policies need to interact with other agents whose real-world behavior might differ from that in the simulation. For example, a self-driving car trained in simulation needs to drive in traffic with human drivers.

Often, the "reality gap" between simulated and real-world agents can be described as a difference in reward functions, e.g., humans might be more risk-averse than AI agents. However, these reality gaps can be hard to precisely quantify, as it is hard to learn the exact reward function of humans. Therefore, robust agents should be robust to uncertainty about the objectives of other agents.

ERMAS is an adversarial robustness solution. ERMAS uses uncertainty sets that describe all "realistic" perturbations of agent reward functions (and hence their resulting behaviors). Consider a self-driving car in traffic. Suppose a Nash equilibrium is for all agents to drive at 70 mph, and a self-driving car has learned in simulation all rational agents would drive at 70 mph. A robust self-driving car needs to account for a situation where a more risk-averse human driver drives at 60 mph, whose 'irrational' behavior is optimal for a reward function that is different from that used in the simulation.

ERMAS's uncertainty set is bounded by the requirement that the simulation's optimal policy be close to optimal even under a perturbed reward function. In our previous example, a "realistic" perturbation of the driver's risk aversion is one where driving at 70mph is not preferable but also not unthinkable. Formally, this upper-bounds the statistical regret of driving at 70mph by some value 'epsilon', where the regret of a policy is defined by subtracting the policy's reward from the optimal policy's reward.

The primary technical challenge of ERMAS is efficiently solving for the worst-case perturbation in this bounded uncertainty set. It does this by dualizing the robustness objective, yielding an optimization objective similar to constrained reinforcement learning. However, optimizing for this objective is difficult as it requires knowing whether a given agent policy is in the uncertainty set. Following the definitions, this requires us to estimate the statistical regret of agent policies which ERMAS avoids by instead estimating regret only within a local region of the policy space. In our previous example, ERMAS estimates the "realism" of a perturbation by only comparing driving at 70mph with the options of driving between 65-75mph, rather than all possible values from 0-100mph.