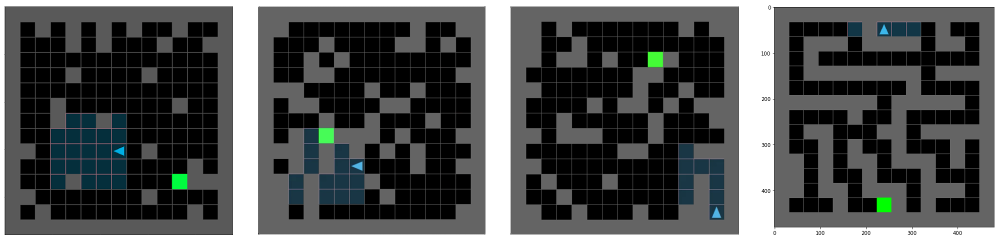

(a) Domain Randomization (b) Minimax Adversarial (c) PAIRED (ours) (d) Transfer task

Figure 1: An agent learns to navigate an environment where the position of the goal and obstacles is an underspecified parameter. If trained using domain randomization to randomly choose the obstacle locations (a), the agent will fail to generalize to a specific or complex configuration of obstacles, such as a maze (d). Minimax adversarial training encourages the adversary to create impossible environments, as shown in (b). In contrast, Protagonist Antagonist Induced Regret Environment Design (PAIRED), trains the adversary based on the difference between the reward of the agent (protagonist) and the reward of a second, antagonist agent. Because the two agents are learning together, the adversary is motivated to create a curriculum of difficult but achievable environments tailored to the agents' current level of performance (c). PAIRED facilitates learning complex behaviors and policies that perform well under zero-shot transfer to challenging new environments at test time.

In order for this automation to be useful, there must be a way of specifying the domain of environments in which the policy should be trained, without needing to fully specify the distribution. In our approach, developers only need to supply an *underspecified environment*: an environment which has free parameters which control its features and behavior. For instance, the developer could give a navigation environment in which the agent's objective is to navigate to the goal and the free parameters are the positions of obstacles. Our method will then construct distributions of environments by providing a distribution of settings of these free parameters; in this case, positions for those blocks. We call this problem of taking the underspecified environment and a policy, and producing an interesting distribution of fully specified environments in which that policy can be further trained *Unsupervised Environment Design* (UED). We formalize unsupervised environment design in Section 3, providing a characterization of the space of possible approaches which subsumes prior work. After training a policy in a distribution of environments generated by UED, we arrive at an updated policy and can use UED to generate more environments in which the updated policy can be trained. In this way, an approach for UED naturally gives rise to an approach for Unsupervised Curriculum Design (UCD). This method can be used to generate capable policies through increasingly complex environments targeted at the policy's current abilities.

Two prior approaches to UED are domain randomization, which generates fully specified environments uniformly randomly regardless of the current policy (e.g., [17, 32, 41]), and adversarially generating environments to minimize the reward of the current policy; *i.e.* minimax training (e.g., [28, 25, 43, 18]). While each of these approaches have their place, they can each fail to generate any interesting environments. In Figure 1 we show examples of maze navigation environments generated by each of these techniques. Uniformly random environments will often fail to generate interesting structures; in the maze example, it will be unlikely to generate walls (Figure 1a). On the other extreme, a minimax adversary is incentivized to make the environments completely unsolvable, generating mazes with unreachable goals (Figure 1b). In many environments, both of these methods fail to generate structured and solvable environments. We present a middle ground, generating environments which maximize regret, which produces difficult but solvable tasks (Figure 1c). Our results show that optimizing regret results in agents that are able to perform difficult transfer task (Figure 1d), which are not possible using the other two techniques.

We propose a novel adversarial training technique which naturally solves the problem of the adversary generating unsolvable environments by introducing an *antagonist* which is allied with the *environment-generating adversary*. For the sake of clarity, we refer to the primary agent we are trying to train as the *protagonist*. The environment adversary's goal is to design environments in which the antagonist achieves high reward and the protagonist receives low reward. If the adversary generates unsolvable environments, the antagonist and protagonist would perform the same and the adversary would get a score of zero, but if the adversary finds environments the antagonist solves and the protagonist does not solve, the adversary achieves a positive score. Thus, the environment adversary is incentivized to more recently, through the use of adversarial RL training [28, 43, 25]. Unlike our algorithm, minimax adversaries have no incentive to guide the learning progress of the agent and can make environments arbitrarily hard or unsolvable. Ghavamzadeh et al. [12] minimize the regret of a model-based policy against a safe baseline to ensure safe policy improvement. Regan and Boutilier [29, 30] study Markov Decision Processes (MDPs) in which the reward function is not fully specified, and use minimax regret as an objective to guide the elicitation of user preferences. While these approaches use similar theoretical techniques to ours, they use analytical solution methods that do not scale to the type of deep learning approach used in this paper, do not attempt to learn to generate environments, and do not consider automatic curriculum generation.

Domain randomization (DR) [17] is an alternative approach in which a designer specifies a set of parameters to which the policy should be robust [32, 41, 40]. These parameters are then randomly sampled for each episode, and a policy is trained that performs well on average across parameter values. However, this does not guarantee good performance on a specific, challenging configuration of parameters. While DR has had empirical success [1], it requires careful parameterization and does not automatically tailor generated environments to the current proficiency of the learning agent. Mehta et al. [23] propose to enhance DR by learning which parameter values lead to the biggest decrease in performance compared to a reference environment.

3 Unsupervised Environment Design

The goal of this work is to construct a policy that performs well across a large set of environments. We train policies by starting with an initially random policy, generating environments based on that policy to best suit its continued learning, train that policy in the generated environments, and repeat until the policy converges or we run out of resources. We then test the trained policies in a set of challenging transfer tasks not provided during training. In this section, we will focus on the environment generation step, which we call *unsupervised environment design* (UED). We will formally define UED as the problem of using an underspecified environment to produce a distribution over fully specified environments, which supports the continued learning of a particular policy. To do this, we must formally define fully specified and underspecified environments, and describe how to create a distribution of environments using UED. We will end this section by proposing minimax regret as a novel approach to UED.

We will model our fully specified environments with a Partially Observable Markov Decision Process (POMDP), which is a tuple $\langle A, O, S, \mathcal{T}, \mathcal{I}, \mathcal{R}, \gamma \rangle$ where A is a set of actions, O is a set of observations, S is a set of states, $\mathcal{T} : S \times A \rightarrow \mathbf{\Delta}(S)$ is a transition function, $\mathcal{I} : S \rightarrow O$ is an observation (or inspection) function, $\mathcal{R} : S \rightarrow \mathbb{R}$, and γ is a discount factor. We will define the utility as $U(\pi) = \sum_{i=0}^{T} r_t \gamma^t$, where T is a horizon.

To model an underspecified environment, we propose the Underspecified Partially Observable Markov Decision Process (UPOMDP) as a tuple $\mathcal{M} = \langle A, O, \Theta, S^{\mathcal{M}}, \mathcal{T}^{\mathcal{M}}, \mathcal{I}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \gamma \rangle$. The only difference between a POMDP and a UPOMDP is that a UPOMDP has a set Θ representing the free parameters of the environment, which can be chosen to be distinct at each time step and are incorporated into the transition function as $\mathcal{T}^{\mathcal{M}} : S \times A \times \Theta \rightarrow \mathbf{\Delta}(S)$. Thus a possible setting of the environment is given by some trajectory of environment parameters $\vec{\theta}$. As an example UPOMDP, consider a simulation of a robot in which $\vec{\theta}$ are additional forces which can be applied at each time step. A setting of the environment $\vec{\theta}$ can be naturally combined with the underspecified environment $\mathcal{M}$ to give a specific POMDP, which we will denote $\mathcal{M}_{\vec{\theta}}$.

In UED, we want to generate a distribution of environments in which to continue training a policy. We would like to curate this distribution of environments to best support the continued learning of a particular agent policy. As such, we can propose a solution to UED by specifying some *environment policy*, $\Lambda : \Pi \rightarrow \mathbf{\Delta}(\Theta^T)$ where Π is the set of possible policies and Θ^T is the set of possible sequences of environment parameters. In Table 2, we see a few examples of possible choices for environment policies, as well as how they correspond to previous literature. Each of these choices also have a corresponding decision rule used for making *decisions under ignorance* [27]. Each decision rule can be understood as a method for choosing a policy given an underspecified environment. This connection between UED and decisions under ignorance extends further than these few decision rules. In the Appendix we will make this connection concrete and show that, under reasonable assumptions, UED and decisions under ignorance are solving the same problem.