Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning

Shariq Iqbal[1] , **Fei Sha** [1,2]

[1] University of Southern California
[2] Google AI

Abstract

Solving tasks with sparse rewards is one of the most important challenges in reinforcement learning. In the single-agent setting, this challenge is addressed by introducing intrinsic rewards that motivate agents to explore unseen regions of their state spaces; however, applying these techniques naively to the multi-agent setting results in agents exploring independently, without any coordination among themselves. Exploration in cooperative multi-agent settings can be accelerated and improved if agents coordinate their exploration. In this paper we introduce a framework for designing intrinsic rewards which consider what other agents have explored such that the agents can coordinate. Then, we develop an approach for learning how to dynamically select between several exploration modalities to maximize extrinsic rewards. Concretely, we formulate the approach as a hierarchical policy where a high-level controller selects among sets of policies trained on diverse intrinsic rewards and the low-level controllers learn the action policies of all agents under these specific rewards. We demonstrate the effectiveness of the proposed approach in cooperative domains with sparse rewards where state-of-the-art methods fail and challenging multi-stage tasks that necessitate changing modes of coordination.

1 Introduction

Solving tasks with sparse rewards is a fundamental challenge of reinforcement learning. This challenge is most commonly addressed by learning with intrinsic rewards that encourage exploration of the state space [Pathak *et al.*, 2017; Houthooft *et al.*, 2016; Burda *et al.*, 2019; Ostrovski *et al.*, 2017; Tang *et al.*, 2017]. In the cooperative multi-agent setting, the sparse reward challenge is exacerbated by the need for agents to *coordinate* their exploration. In many cases, the non-coordinated approach – agents exploring independently – is not efficient. For example, consider a search-and-rescue task where multiple agents need to collectively find all missing persons spread throughout their environment and bring them to a common recovery location. During the search

phase, it would be inefficient for the agents to explore the same areas redundantly. Instead, it would be much more sensible for agents to "divide-and-conquer" or avoid redundant exploration. Thus, an ideal intrinsic reward for this phase would encourage such behavior; however, the same behavior would not be ideal during the recovery phase where agents must converge at a common location. Cooperative multi-agent reinforcement learning can benefit from coordinating exploration across agents; however, the type of coordination should be *adaptive* to the task at hand.

In this work, we introduce a framework for designing multi-agent intrinsic rewards that coordinate with respect to explored regions, then we present a method for learning both low-level policies trained on different intrinsic rewards and a meta-policy for selecting the policies which maximize extrinsic rewards on a given task[1]. Importantly, we learn the policies simultaneously using a shared replay buffer with off-policy methods, drastically improving sample efficiency. This shared replay buffer enables us to use *all* data to train *all* policies, rather than needing to collect data with the specific policies we want to update. Moreover, the meta-policy is learned in conjunction with those low-level policies, effectively exploring over the space of coordinated low-level exploration types. We show empirically, in both a Grid-World domain as well as in the more complex 3D ViZDoom [Kempka *et al.*, 2016] setting: 1) intrinsic reward functions which coordinate across agents are more effective than independent intrinsically motivated exploration, 2) our approach is able to match or exceed the performance of the best coordinated intrinsic reward function (which differs across tasks) while using no more samples, and 3) on challenging multi-stage tasks requiring varying modes of cooperation, our adaptive approach outperforms all individual reward functions and continues to learn while the other approaches stagnate due to their lack of coordination and/or adaptability.

2 Related Work

Single-Agent Exploration In order to solve sparse reward problems, researchers have long worked on improving exploration in reinforcement learning. Prior works commonly propose reward bonuses that encourage agents to reach novel

---

states. In tabular domains, reward bonuses based on the inverse state-action count have been shown to be effective in accelerating learning [Strehl and Littman, 2008]. In order to scale count-based approaches to large state spaces, many recent works have focused on devising pseudo state counts to use as reward bonuses [Bellemare *et al.*, 2016; Ostrovski *et al.*, 2017; Tang *et al.*, 2017]. Alternatively, some work has focused on defining intrinsic rewards for exploration based on inspiration from psychology [Oudeyer and Kaplan, 2009; Schmidhuber, 2010]. These works use various measures of state novelty as intrinsic rewards motivating exploration [Pathak *et al.*, 2017; Houthooft *et al.*, 2016; Burda *et al.*, 2019]

Exploration in MARL Jaques *et al.* [2018] define an intrinsic reward function for multi-agent reinforcement learning that encourages agents to take actions which have the biggest effect on other agents' behavior, otherwise referred to as "social influence". Agogino and Tumer [2008] defines metrics for evaluating the efficacy of reward functions in multi-agent domains. These works, while important, do not address the problem of coordinating exploration in a large state space among multiple agents. A recent approach to collaborative evolutionary reinforcement learning [Khadka *et al.*, 2019] shares some similarities with our approach. As in our work, the authors devise a method for learning a population of diverse policies with a shared replay buffer and dynamically selecting the best learner; however, their work is focused on single-agent tasks and does not incorporate any notion of intrinsic rewards for exploration. Wang *et al.* [2020] define influence-based rewards which encourage agents to visit regions where their actions *influence* other agents' transitions and rewards (e.g. one agent unlocks a door for another). In practice, they combine their approach with state-based exploration, and thus, their work is complementary and orthogonal to our approach where agents do not simply explore regions that are novel to them but take into account how novel all other agents consider these regions. This work is most applicable to settings where agents influencing each others' dynamics forms a positive inductive bias towards solving a task. Most recently, Mahajan *et al.* [2019] introduce a mechanism for achieving "committed" exploration, allowing agents to explore temporally extended coordinated strategies. While this approach enables coordinated exploration, it does not encourage exploration of novel states, and as such, may not learn effectively in sparse reward tasks.

3 Background

Dec-POMDPs In this work, we consider the setting of decentralized POMDPs [Oliehoek *et al.*, 2016], which are used to describe cooperative multi-agent tasks. A decentralized POMDP (Dec-POMDP) is defined by a tuple: $(\mathbf{S}, \mathbf{A}, T, \mathbf{O}, O, R, n, \gamma)$. In this setting we have n total agents. $\mathbf{S}$ is the set of global states in the environment, while $\mathbf{O} = \otimes_{i \in \{1...n\}} \mathbf{O}_i$ is the set of joint observations for each agent and $\mathbf{A} = \otimes_{i \in \{1...n\}} \mathbf{A}_i$ is the set of possible joint actions for each agent. A specific joint action at one time step is denoted as $\mathbf{a} = \{a_1, \ldots, a_n\} \in \mathbf{A}$ and a joint observation is $\mathbf{o} = \{o_1, \ldots, o_n\} \in \mathbf{O}$. T is the state transition function

which defines the probability $P(s'|s, \mathbf{a})$, and O is the observation function which defines the probability $P(\mathbf{o}|\mathbf{a}, s')$. R is the reward function which maps the combination of state and joint actions to a single scalar reward. Importantly, this reward is shared between all agents, so Dec-POMDPs always describe cooperative problems. Finally, γ is the discount factor which determines how much the agents should favor immediate reward over long-term gain.

Soft Actor-Critic Our approach uses Soft Actor-Critic (SAC) [Haarnoja *et al.*, 2018] as its underlying algorithm. SAC incorporates an entropy term in the loss functions for both the actor and critic, in order to encourage exploration and prevent premature convergence to a sub-optimal deterministic policy. The policy gradient with an entropy term is computed as follows:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim D, a \sim \pi} \left[\nabla_\theta \log \pi_\theta(a|s) \left(-\frac{\log \pi_\theta(a|s)}{\alpha} + Q_\psi(s, a) - b(s) \right) \right] \quad (1)$$

where D is a replay buffer that stores past environment transitions, ψ are the parameters of the learned critic, $b(s)$ is a state dependent baseline (e.g. the state value function $V(s)$), and α is a reward scale parameter determining the amount of entropy in an optimal policy. The critic is learned with the following loss function:

$$\mathcal{L}_Q(\psi) = \mathbb{E}_{(s,a,r,s') \sim D} \left[(Q_\psi(s, a) - y)^2 \right] \quad (2)$$

$$y = r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} \left[Q_{\bar{\psi}}(s', a') - \frac{\log(\pi_{\bar{\theta}}(a'|s'))}{\alpha} \right] \quad (3)$$

where $\bar{\psi}$ are the parameters of the target critic which is an exponential moving average of the past critics, updated as: $\bar{\psi} \leftarrow (1 - \tau)\bar{\psi} + \tau\psi$, and τ is a hyperparameter that controls the update rate.

Centralized Training with Decentralized Execution A number of works in deep multi-agent reinforcement learning have followed the paradigm of centralized training with decentralized execution [Lowe *et al.*, 2017; Foerster *et al.*, 2018; Sunehag *et al.*, 2018; Rashid *et al.*, 2018; Iqbal and Sha, 2019]. This paradigm allows for agents to train while sharing information (or incorporating information that is unavailable at test time) but act using only local information, without requiring communication which may be costly upon execution. Since most reinforcement learning applications use simulation for training, communication between agents during the training phase has a relatively low cost.

4 Intrinsic Rewards for MARL Exploration

In this section we describe intrinsic reward functions for exploration that are specifically tailored to multi-agent learning. The main idea is to share whether other agents have explored a region and consider it as novel.

We assume that each agent (indexed by i) has a novelty function $f_i : \mathbf{O}_i \to \mathbb{R}^+$ that determines how novel an observation is to it, based on its past experience. This function can be an inverse state visit count in discrete domains,
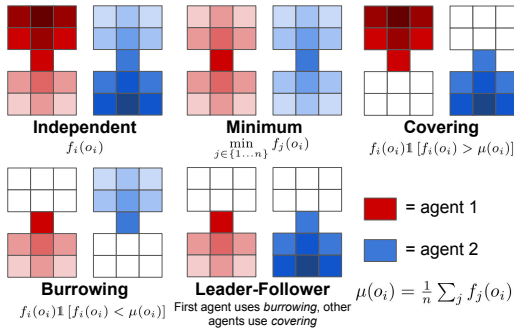
Figure 1: Multi-agent intrinsic rewards. Visualized for the 2 agent case. INDEPENDENT shows the regions that have been explored by each agent. Darker shades mean higher reward values.

or, in large/continuous domains, it can be represented by recent approaches for developing novelty-based intrinsic rewards in complex domains, such as random network distillation [Burda *et al.*, 2019]. We assume that all agents share the same observation space so that each agent's novelty function can operate on all other agents' observations.

We define the multi-agent intrinsic reward function $g_i(\cdot)$ for agent i, which considers how novel *all* agents consider i's observation. Concretely, the function maps the vector $[f_1(o_i), \cdots, f_n(o_i)]$ to a scalar reward.

Desiderata of $g_i(\cdot)$ While in theory $g_i(\cdot)$ can be in any form, we believe the following two properties are intuitive and naturally applicable to cooperative MARL:

- **Coordinate-wise Monotonicity** An observation becoming less novel to any individual agent should *not increase* the intrinsic reward, preventing the agent from exploring a region more as it becomes more known to another agent. Formally, $\partial g_i / \partial f_j \geq 0, \forall\, i, j$.

- **Inner-directedness** If an observation approaches having zero novelty to an agent, then the intrinsic reward should also approach zero, irrespective of other agents' novelty. This prevents the agent from repetitively exploring the same regions, at the "persuasion" of other agents.

Examples Fig. 1 visualizes examples of intrinsic rewards that observe the aforementioned desirable properties. INDEPENDENT rewards are analogous to single-agent approaches to exploration which define the intrinsic reward for an agent as the novelty of their own observation that occurs as a result of an action. The remainder of intrinsic reward functions that we consider use the novelty functions of other agents, in addition to their own, to further inform their exploration.

MINIMUM rewards consider how novel all agents find a specific agent's observation and rewards that agent based on the minimum. This method leads to agents only being rewarded for exploring areas that no other agent has explored, which could be advantageous in scenarios where redundancy in exploration is not useful or even harmful. COVERING rewards agents for exploring areas that it considers more novel than the average agent. This reward results in agents shifting around the state space, only exploring regions as long as they

are more novel to them than their average teammate. BURROWING rewards do the opposite, only rewarding agents for exploring areas that it considers less novel than the average agent. While seemingly counterintuitive, these rewards encourage agents to further explore areas they have already explored with the hope that they will discover new regions that few or no other agents have seen, which they will then consider less novel than average and continue to explore. As such, these rewards result in agents continuing to explore until they exhaust all possible intrinsic rewards from a given region (i.e. hit a dead end), somewhat akin to a depth-first search. LEADER-FOLLOWER uses burrowing rewards for the first agent, and covering rewards for the rest of the agents. This leads to an agent exploring a space thoroughly, and the rest of the agents following along and trying to cover that space.

Note that these are not meant to be a comprehensive set of intrinsic reward functions applicable to all cooperative multi-agent tasks. In fact, any convex combination of them is consistent with the desiderata. We use the set of rewards introduced in Fig. 1 to concurrently learn a set of diverse exploration policies and introduce in the next section a method for dynamically switching between them. This method matches or exceeds the performance of the best individual exploration method in several complex tasks.

5 Learning for Multi-Agent Exploration

For many tasks, it is impossible to know a priori what type of exploration may provide the right inductive bias. Furthermore, the type that is most helpful could change over the course of training if the task is sufficiently complex. In this section we present our approach for simultaneously learning policies trained with different types of intrinsic rewards and dynamically selecting the best one.

Simultaneous Policy Learning In order to learn policies for various types of intrinsic rewards in parallel, we utilize a shared replay buffer and off-policy learning to maximize sample efficiency. In other words, we learn policies and value functions for *all* intrinsic reward types from *all* collected data, regardless of which policies it was collected by. This parallel learning is made possible by the fact that we can compute our novelty functions off-policy, given the observations for each agent after each environment transition, which are saved in a replay buffer. In Figure 2 we visualize our model architecture. We share a critic base and split extrinsic and intrinsic return heads as in Burda *et al.* [2019]. We learn separate heads for each agent $i \in \{1 \ldots n\}$ and reward $j \in \{1 \ldots m\}$ where m is the total number of intrinsic reward types that we are considering. For policies, each agent learns its own base that is shared across separate heads for all intrinsic reward types. Our specific learning algorithm is adapted from the multi-agent Soft-Actor-Critic method presented in Iqbal and Sha [2019].

The policy for agent i, trained using reward j (in addition to extrinsic rewards), is represented by π_i^j. The parameters of this policy are $\Theta_i^j = \{\theta_i^{\text{share}}, \theta_i^j\}$, where θ_i^{share} is the shared base/input (for agent i) and θ_i^j is a head/output specific to
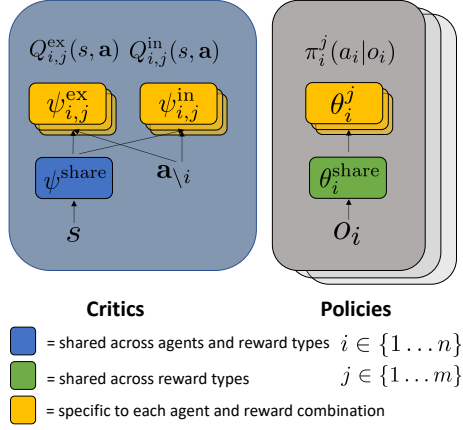
Figure 2: Diagram of our model architecture. Colors indicate how parameters are shared. i indexes agents, while j indexes reward types. Full architectures in pseudo-code are provided in the Supp.

this reward type. The extrinsic critic for policy head π_i^j is represented by $Q_{i,j}^{\text{ex}}$. It takes as input the global state s and the actions of all other agents $\mathbf{a}_{\setminus i}$, and it outputs the expected returns under policy π_i^j for each possible action that agent i can take, given all other agents' actions. The parameters of this critic are $\Psi_{i,j}^{\text{ex}} = \{\psi^{\text{share}}, \psi_{i,j}^{\text{ex}}\}$ where ψ^{share} is a shared base across all agents and reward types. A critic with similar structure exists for predicting the intrinsic returns of actions taken by π_i^j, represented by $Q_{i,j}^{\text{in}}$, which uses the parameters: $\Psi_{i,j}^{\text{in}} = \{\psi^{\text{share}}, \psi_{i,j}^{\text{in}}\}$. Note that the intrinsic critics share the same base parameters ψ^{share}.

In our notation, we use the absence of a subscript or superscript to refer to a group. For example π^j, refers to *all* agents' policies trained on intrinsic reward j. We train our critics with the soft actor-critic Q-function loss (Equation 2), using the following target, $y_{i,j}^{\text{ex/in}}$, for the extrinsic and intrinsic critics:

$$r^{\text{ex/in}} + \gamma \mathbb{E}_{\mathbf{a}' \sim \bar{\pi}^j} \left[\bar{Q}_{i,j}^{\text{ex/in}}(s', \mathbf{a}') - \frac{\log(\bar{\pi}_i^j(a_i'|o_i'))}{\alpha} \right] \quad (4)$$

The extrinsic target, $y_{i,j}^{\text{ex}}$, uses the Dec-POMDP's reward function $r^{\text{ex}}(s, \mathbf{a})$, while the intrinsic target, $y_{i,j}^{\text{in}}$ uses the intrinsic reward $r_{i,j}^{\text{in}}(o_i')$. Note that the intrinsic rewards depend on the observation resulting from the actions taken, o_i'. $\bar{Q}_{i,j}^{\text{ex/in}}$ refers to the target Q-function, an exponential weighted average of the past Q-functions, used for stability. $\bar{\pi}_i^j$ are similarly updated target policies. We train each policy head with the following gradient:

$$\nabla_{\Theta_i^j} J(\pi_i^j) = \mathbb{E}_{(s, \mathbf{o}) \sim D, \mathbf{a} \sim \pi^j} \Big[\nabla_{\Theta_i^j} \log \pi_i^j(a_i|o_i) \\ \left(-\frac{\log \pi_i^j(a_i|o_i)}{\alpha} + A_i^j(s, \mathbf{a}) \right) \Big] \quad (5)$$

$$A_i^j(s, \mathbf{a}) = Q_{i,j}^{\text{ex}}(s, \mathbf{a}) + \beta Q_{i,j}^{\text{in}}(s, \mathbf{a}) - V_i^j(s, \mathbf{a}_{\setminus i}) \quad (6)$$

$$V_i^j(s, \mathbf{a}_{\setminus i}) = \sum_{a_i' \in \mathbf{A}_i} \pi_i^j(a_i'|o_i)(Q_{i,j}^{\text{ex}}(s, \{a_i', \mathbf{a}_{\setminus i}\}) + \\ \beta Q_{i,j}^{\text{in}}(s, \{a_i', \mathbf{a}_{\setminus i}\})) \quad (7)$$

where β is a scalar that determines the weight of the intrinsic rewards, relative to extrinsic rewards, and A_i^j is a multi-agent advantage function [Foerster *et al.*, 2018; Iqbal and Sha, 2019], used for helping with multi-agent credit assignment. We update *all* policy and critic heads with each environment transition sample.

Dynamic Policy Selection Now that we have established a method for simultaneously learning policies using different intrinsic reward types, we must devise a means of selecting between these policies when collecting data in the environment (i.e., rollouts). In order to select policies to use for rollouts, we must consider which policies maximize extrinsic returns, while taking into account the fact that there may still be "unknown unknowns," or regions that the agents have not seen yet where they may be able to further increase their extrinsic returns. As such, we must learn a meta-policy that selects from the set of policies trained on different intrinsic rewards to maximize extrinsic returns while maintaining some degree of stochasticity. We parameterize the selector policy Π with a vector, ϕ, that contains an entry for every reward type. The probability of sampling head j is: $\Pi(j) \propto \exp(\phi[j])$. We sample from the meta-policy at the start of each rollout. Using policy gradients, we train the policy selector, Π, to maximize extrinsic returns:

$$\nabla_\phi J(\Pi) = \mathbb{E}_{h \sim \Pi} \Big[\nabla_\phi \log \Pi(h) \\ \left(-\frac{\log \Pi(h)}{\eta} + R_h^{\text{ex}} - b_\Pi \right) \Big] \quad (8)$$

$$R_h^{\text{ex}} = \sum_{t=0}^{T} \gamma^t r^{\text{ex}}(s_t, \mathbf{a}_t) | \mathbf{a} \sim \pi^h, \quad b_\Pi = \sum_{h'}^{m} \Pi(h')\mu_{h'} \quad (9)$$

μ_h is a running mean of the returns received by head h in the past, and η is a parameter similar to α for the low-level policies, which promotes entropy in the selector policy. Entropy in the policy selector prevents it from collapsing onto a single exploration type that exploits a local optimum.

6 Experiments

We first train policies with each intrinsic reward function defined in Fig. 1, then compare our approach as well as several baselines and ablations. We refer to the best performing reward type for each setting as the "non-adaptive oracle". "Non-adaptive" is used to contrast with our approach which can adapt different exploration strategies during training, while "oracle" is used since we do not know a priori which type will perform best. In our experiments we validate the following hypotheses: 1) Multi-agent intrinsic reward functions improve performance on tasks requiring coordination, 2) Our approach matches the performance of the non-adaptive oracle without training separate policies, and 3) In tasks requiring changing coordination strategies, our method outperforms the non-adaptive oracle.
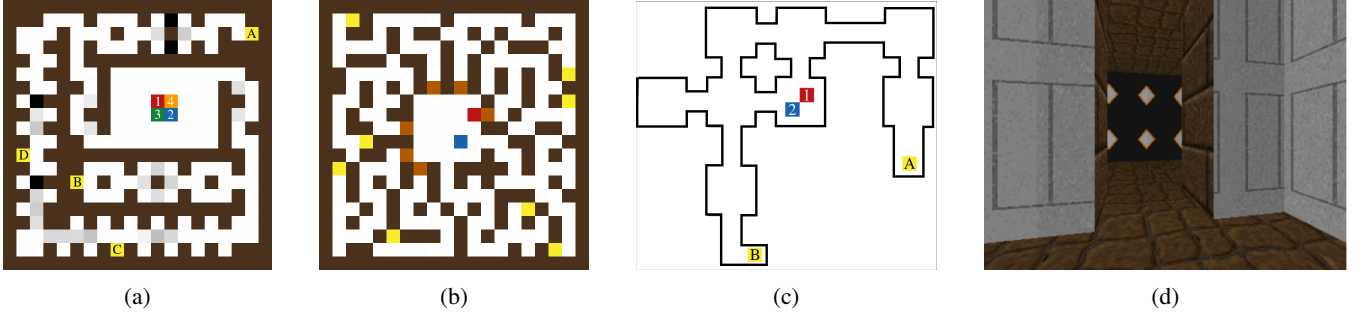
Figure 3: (a) Rendering of the fixed map in our gridworld domain. (b) Randomly generated map used for **FLIP-TASK** task. n doors to new paths open every time either **TASK 1** or **TASK 2** is completed. (c) Top-Down view of VizDoom "My Way Home" map, modified for multi-agent experiments. (d) Egocentric view in VizDoom used for agents' observations.

6.1 Tasks

Tasks for testing single-agent exploration typically revolve around navigation of an environment with sparsely distributed rewards (e.g. Montezuma's Revenge [Ostrovski *et al.*, 2017; Tang *et al.*, 2017; Burda *et al.*, 2019], Viz-Doom [Pathak *et al.*, 2017], etc). In the multi-agent setting, we define tasks which similarly consider navigation with very sparse rewards, while requiring varying modalities of coordination across agents. These tasks involve collecting the items spread around a map (displayed in yellow in Figure 3):

TASK 1: Agents must cooperatively collect *all* treasure on the map in order to complete the task. Ideally, agents should spread out in order to solve the task effectively. **TASK 2:** Agents must all collect the *same* treasure. Thus, agents would ideally explore similar regions concurrently. **TASK 3:** Agents must all collect the specific treasure assigned to them, requiring no coordination across agents.

TASK 1 and **TASK 2** need to solve coordination problems, as an individual agent can repeat the same behavior and receive drastically different returns depending on the behavior of other agents. **TASK 3** is intended as a sanity check where independent exploration should perform best. All 3 tasks are tested on the maps pictured in Figures 3a and 3c.

FLIP-TASK is a task where the modality of required coordination changes as agents progress, akin to the search and rescue task mentioned in the introduction. This task is tested on randomly generated maps, an example of which is pictured in Figure 3b. In **FLIP-TASK** agents begin in a central room with n branching paths available (where treasures are placed at the furthest available point) and must solve either **TASK 1** or **TASK 2** with respect to the available treasures. Once this task is complete, the next set of n paths (blocked by the light brown doors) opens up for which the task will be the opposite of the previous task ($1 \rightarrow 2, 2 \rightarrow 1$), requiring agents to adapt their exploration strategy after they learn to solve the first task.

Agents receive a negative time penalty in their extrinsic rewards at each step, so they are motivated to complete the task as quickly as possible. The only positive extrinsic reward comes from any agent collecting a treasure allowed by the specific task, and rewards are shared between all agents.

6.2 Domains

We first test our approach using a multi-agent gridworld domain (pictured in Fig. 3a and 3b). Then, in order to test our method's ability to scale to more complex 3D environments with visual observations, we test on the VizDoom framework [Kempka *et al.*, 2016].

The novelty function for each agent f_i, which is used for calculating the intrinsic rewards in Figure 1, is defined as $\frac{1}{N^\zeta}$, where N is the number of times the agent has visited its current cell and ζ is a decay rate selected as a hyperparameter (we find $\zeta = 0.7$ works well for our purposes). Since Viz-Doom is not a discrete domain, we discretize agents' (x, y) positions into bins and use the counts for these bins.

6.3 Baselines and Ablations

We consider another approach to adapting single-agent intrinsic rewards to MARL in *Centralized*, where we provide intrinsic rewards to all agents as if they were a single agent. In other words, we use the inverse count of the number of times *all* agents have jointly taken up their combined positions. We also evaluate QMIX [Rashid *et al.*, 2018], a state-of-the-art method in cooperative MARL, and MAVEN [Mahajan *et al.*, 2019], which builds on QMIX by incorporating committed temporally extended exploration. We use the authors' open-sourced code for these comparisons.

Finally, we conduct ablation studies to determine the effectiveness of our meta-policy in balancing exploration and exploitation. *Multi (Uniform Meta-Policy)* samples action policies at random and *Multi (No Entropy)* does not incorporate entropy to encourage exploration. We also tested the base multi-agent SAC algorithm without intrinsic rewards (MA-SAC).

6.4 Results

Results (non-adaptive oracle and our approach) in all settings are summarized in Table 1, and several training curves are found in Figure 4. The detailed training curves and a table of final performance for all intrinsic reward types in all settings can be found in the supplementary material. We note the type of the non-adaptive oracle is frequently not INDEPENDENT, indicating that the multi-agent intrinsic rewards introduced in section 4 are effective in comparison to a naive application of single agent intrisic reward methods. We also include in the supplement a discussion of the behavior of each individual reward type along with videos demonstrating this behavior.

GRIDWORLD				
Task	n	NAO Type	NAO	MULTI
1	2	BURROWING	1.98 ± 0.06	2.00 ± 0.00
	3	BURROWING	2.06 ± 1.05	2.23 ± 0.73
	4	BURROWING	1.90 ± 0.49	2.04 ± 0.61
2	2	INDEPENDENT	2.00 ± 0.00	1.83 ± 0.41
	3	LEAD-FOLLOW	3.00 ± 0.00	1.80 ± 0.71
	4	LEAD-FOLLOW	2.66 ± 2.06	2.54 ± 1.21
3	2	INDEPENDENT	1.39 ± 0.94	2.00 ± 0.00
	3	INDEPENDENT	1.68 ± 0.70	2.21 ± 0.91
	4	BURROWING	2.14 ± 1.49	1.73 ± 0.47
Flip-Task	2	INDEPENDENT	3.04 ± 1.33	4.03 ± 0.97
VIZDOOM				
1	2	BURROWING	1.94 ± 0.10	1.98 ± 0.03
2	2	LEAD-FOLLOW	1.93 ± 0.10	1.23 ± 0.65
3	2	MINIMUM	0.64 ± 1.05	1.64 ± 0.63

Table 1: # of treasures found with standard deviation across 6 runs. Our method (**MULTI**) matches or outperforms the non-adaptive oracle (NAO) in nearly all settings
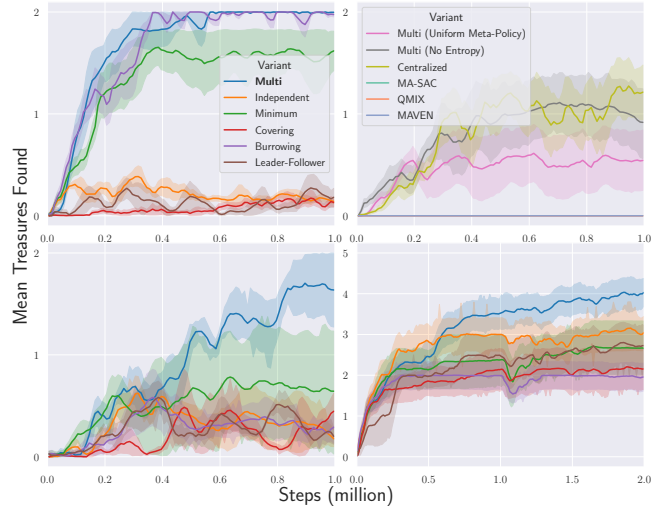

Figure 4: (Top Left) Mean number of treasures found per episode on **TASK 1** with 2 agents in the gridworld domain. (Top Right) Ablations and baselines in the same setting. (Bottom Left) **TASK 3** with 2 agents on VizDoom. (Bottom Right) **FLIP-TASK** with 2 agents on gridworld. Shaded regions are a 68% confidence interval across 6 runs of the running mean over the past 100 episodes.

Matching the Oracle Performance with a Meta-Policy We find our approach is competitive with the non-adaptive oracle in nearly all tasks, while only needing the same number of samples as a *single* run (a more fair comparison would allow our method to train for the sum of samples used to identify the oracle). This performance is exciting, as our method receives no prior information about the optimal type of exploration, while each type carries its own bias. Furthermore, we find our results on the more complex VizDoom domain mirror those in the gridworld, indicating our methods are not limited to discrete domains, assuming a reliable way for measuring the novelty of observations exists.

Advantages of Adaptive Strategies In two cases we find our approach is able to surpass the performance of the non-adaptive oracle: **TASK 3** and **FLIP-TASK**. In the case of **TASK 3**, this is interesting as rewards for each agent are assigned independently, so coordination is not strictly necessary. In this case, multi-agent intrinsic rewards introduce useful biases that naturally divide the space into explorable regions for each agent, reducing the chances of "detachment" [Ecoffet *et al.*, 2019] (which we find independent exploration to suffer from); however, in this task they may divide the space into the *wrong* regions for each agent to get their rewards. Our approach succeeds due to its ability to reap the benefits of multi-agent intrinsic rewards while being able to switch strategies if one is not working. We analyze this meta-policy behavior in detail in the supplementary material. The success on **TASK 3** suggests that our approach may be beneficial in the single-agent setting when potential paths of exloration are numerous (e.g. by training several policies for one task and treating them as separate agents in our reward functions).

Our approach is similarly successful on **FLIP-TASK**. Due to the adaptability of our meta-policy, the agents are able to switch their exploration type to suit the next task after they learn to solve the first one. Note that all sets of exploration policies will learn to solve the first task once any single one does since they share experience and are trained on a combination of extrinsic and intrinsic rewards. As such, the meta-

policy can switch to an exploration strategy best suited for the next task, knowing that it will reliably solve the previous task.

The Unexpected Challenge of TASK 2 We find our approach is unable to match the performance of the non-adaptive oracle on **TASK 2** in certain cases (gridworld with 3 agents and VizDoom). This lack of success may be an indication that the exploration strategies which perform well in these settings require commitment to a single strategy early on in training, highlighting a limitation of our approach. Our method requires testing out all policies until we find one that reaches high extrinsic rewards, which can dilute the effectiveness of exploration early on.

Ablations and Baselines We test against several baselines in the gridworld setting on **TASK 1** with 2 agents (top right of Fig. 4). Training curves for other tasks can be found in the Supplement. We find that those results show the same patterns as **TASK 1**. We find that our approach balances meta-exploration and exploitation by outperforming both *Multi (Uniform Meta-Policy)* (pure explore) and *Multi (No Entropy)* (pure exploit). While *Centralized* will ensure the global state space is thoroughly searched, it lacks the inductive biases toward spatial coordination that our reward functions incorporate. As such, it does not learn as efficiently as our method. Finally, all three of MA-SAC, QMIX, and MAVEN fail to learn in our setting. None of these incorporate novelty-seeking exploration, which are crucial in sparse reward domains.

7 Conclusion

We propose a framework for designing multi-agent intrinsic reward functions with diverse properties, and compare an varied set on several multi-agent exploration tasks in a gridworld domain as well as in VizDoom. Overall, we can see that cooperative multi-agent tasks can, in many cases, benefit from

intrinsic rewards that take into account what other agents have explored, but there are various ways to incorporate that information, each resulting in different coordinated behaviors. We show that our method is capable of matching or surpassing the performance of the non-adaptive oracle on various tasks while using the same number of samples collected from the environment. Furthermore, we show that adaptation of exploration type over the course of training can overcome the limitations of choosing a fixed exploration type.

References

Adrian K Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, 10–15 Jul 2018.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970. PMLR, 09–15 Jun 2019.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A Ortega, D J Strouse, Joel Z Leibo, and Nando de Freitas. Social influence as intrinsic motivation for Multi-Agent deep reinforcement learning. *arXiv preprint arXiv:1810.08647*, October 2018.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learn-ing. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348. IEEE, Sep 2016.

Shauharda Khadka, Somdeb Majumdar, Santiago Miret, Evren Tumer, Tarek Nassar, Zach Dwiel, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. *arXiv preprint arXiv:1905.00976*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6382–6393, 2017.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, pages 7613–7624, 2019.

Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 06–11 Aug 2017.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304, 10–15 Jul 2018.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309 – 1331, 2008. Learning Theory 2005.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on*

Autonomous Agents and MultiAgent Systems, AAMAS '18, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. In *International Conference on Learning Representations*, 2020.

A   Environment Details

A.1   Gridworld

At each step there is a 10% chance of an agent's action being replaced by a random one. Inspired by the enemies in Montezuma's Revenge which increase the challenge of exploration, we introduce "wormholes" which have a probability of opening at each time step. This probability, which agents can observe, changes at each step using a biased random walk such that it moves toward one, until the hole opens and it resets to zero. If an agent steps into an open wormhole, they will be sent back to their starting position. These are an important aspect of the environment, as they add difficulty to exploration. The probability, ρ, of a wormhole opening at each step, t, evolves as such: $\rho_{t+1} = \rho_t + \mathcal{N}(\mu, \sigma)$, where $\mu = \sigma = 0.05$ for **TASK 1** and $\mu = \sigma = 0.005$ for **2** and **3**.

Agents observe their global position in (x, y) coordinates (scalars), as well as local information regarding walls in adjacent spaces, the probability of their adjacent spaces opening into a wormhole, the relative position of other agents (if they are within 3 spaces), as well as information about which treasures the agent has already collected in the given episode. The global state is represented by the (x, y) coordinates of all agents, as one-hot encoded vectors for x and y separately, as well as the local information of all agents regarding wormholes, walls, and treasures collected. Each agent's action space consists of the 4 cardinal directions as well as an option to not move, which is helpful in cases where an agent is waiting for a wormhole to be safe to cross.

A.2   VizDoom

We use the "My Way Home" map, which has been used as a test bed for single-agent exploration techniques [Pathak *et al.*, 2017], and modify it for multi-agent tasks. Since the agents are moved to a central location closer to their rewards than in the original map, we lower the frame skip from 4 to 2, in order to force agents to take twice as many steps in order to explore the same areas, maintaining the challenging nature of exploration in the original task. Agents receive their egocentric view in the form of 48x48 grayscale images as observations along with an indicator of which agents (if any) have collected each reward, and we use a vector based global state which includes all agents' (x, y) positions and velocities, their orientations, as well as the same indicator of which agent has collected each reward. The policies *do not* receive

the global state as input, as it is only used for the centralized critics. As in the gridworld setting, we use count-based intrinsic rewards for VizDoom; however, since VizDoom is not a discrete domain, we separate agents' (x, y) positions into discrete bins and use the counts for these bins. There are 30 bins in the x dimension and 26 in the y dimension. (x, y) positions in the global state are represented both as scalars and one-hot vectors indicating which bin the agents are currently occupying. Each agent can choose from 3 actions at each time step: turn left, turn right, or go forward.

B   Training Details

Algorithm 1 Training Procedure for Multi-Explore w/ Soft Actor-Critic [Haarnoja *et al.*, 2018]

1: Initialize environment with n agents
2: Initialize replay buffer, D
3: $t_{\text{update}} \leftarrow 0$
4: $t_{\text{ep}} \leftarrow$ max ep length
5: **for** $t = 1 \ldots$ total steps **do**
6:    **if** episode done or $t_{\text{ep}} ==$ max ep length **then**
7:       **for** $j = 1 \ldots$ niters-selector **do**
8:          *UpdateSelector*(R, h) {Eqs 8-9 in main text}
9:       **end for**
10:      $s, \mathbf{o} \leftarrow ResetEnv()$
11:      $h \sim \Pi$ {Sample policy head}
12:      $t_{\text{ep}} \leftarrow 0$
13:      $R \leftarrow 0$
14:   **end if**
15:   Select actions $a_i \sim \pi_i^h(\cdot | o_i)$ for each agent, i
16:   Send actions to environment and get $s', \mathbf{o}', r$
17:   $R \leftarrow R + \gamma^{t_{\text{ep}}} r$
18:   Store transition $(s, \mathbf{o}, \mathbf{a}, s', \mathbf{o}', r)$ in D
19:   $t_{\text{update}} + = 1$
20:   $t_{\text{ep}} + = 1$
21:   **if** $t_{\text{update}} ==$ steps per update **then**
22:      **for** $j = 1 \ldots$ niters-model **do**
23:         Sample minibatch, $B \sim D$
24:         *UpdateCritic*(B) {Eqs 2,4 in main text}
25:         *UpdatePolicies*(B) {Eqs 5-7 in main text}
26:         Update target parameters:

$$\bar{\Psi} = \tau\bar{\Psi} + (1 - \tau)\Psi$$

$$\bar{\Theta} = \tau\bar{\Theta} + (1 - \tau)\Theta$$

27:      **end for**
28:      $t_{\text{update}} \leftarrow 0$
29:   **end if**
30: **end for**

The training procedure is detailed in Algorithm 1, and all hyperparameters are listed in Tables 2 and 3. Hyperparameters were selected by tuning one parameter at a time through intuition on task 1 with 2 agents and then applying to the rest of the settings with minimal changes. Individual runs used a single NVIDIA Titan Xp GPU for training. Two machines were used, one (desktop) has an Intel i7-6800K CPU with 128 GB RAM, and another (server) has two Intel Xeon(R) Gold

6154 CPUs with 256 GB RAM. Both machines run Ubuntu 16.04. Software versions are provided in the code supplement.

C Network Architectures

In this section we list, in pseudo-code, the architectures we used for all policies and critics

C.1 Gridworld

θ_i^{share} (shared for policy heads):

```
obs_size = observations.shape[1]
fc1 = Linear(in_dim=obs_size, out_dim=128)
nl1 = ReLU()
```

θ_i^j (specific to each policy head):

```
n_acs = actions.shape[1]
fc2 = Linear(in_dim=fc1.out_dim, out_dim=32)
nl2 = ReLU()
fc3 = Linear(in_dim=fc2.out_sim, out_dim=n_acs)
```

ψ^{share} (shared across critics for all agents and reward types):

```
state_size = states.shape[1]
fc1 = Linear(in_dim=state_size, out_dim=128)
nl1 = ReLU()
```

$\psi_{i,j}$ (specific to each agent/policy head combination, same architecture for extrinsic and intrinsic critics):

```
n_acs = actions.shape[1]
# fc2 takes other agents' actions as input
fc2 = Linear(in_dim=fc1.out_dim + (num_agents - 1)
    * n_acs, out_dim=128)
nl2 = ReLU()
fc3 = Linear(in_dim=fc2.out_dim, out_dim=n_acs)
```

C.2 VizDoom

θ_i^{share} (shared for policy heads belonging to one agent):

```
# vector observation encoder
vect_obs_size = vector_observations.shape[1]
vect_fc = Linear(in_dim=obs_size, out_dim=32)
vect_nl = ReLU()
# image observation encoder
img_obs_channels = image_observations.shape[1]
pad1 = ReflectionPadding(size=1)
conv1 = Conv2D(in_channels=img_obs_channels,
    out_channels=32, filter_size=3, stride=2)
conv_nl1 = ReLU()
pad2 = ReflectionPadding(size=1)
conv2 = Conv2D(in_channels=conv1.out_channels,
    out_channels=32, filter_size=3, stride=2)
conv_nl2 = ReLU()
pad3 = ReflectionPadding(size=1)
conv3 = Conv2D(in_channels=conv2.out_channels,
    out_channels=32, filter_size=3, stride=2)
conv_nl3 = ReLU()
```

```
pad4 = ReflectionPadding(size=1)
conv4 = Conv2D(in_channels=conv3.out_channels,
    out_channels=32, filter_size=3, stride=2)
conv_nl4 = ReLU()
conv_flatten = Flatten() # flatten output of conv
    layers
conv_fc = Linear(in_dim=conv_flatten.out_dim,
    out_dim=128)
conv_fc_nl = ReLU()
```

θ_i^j (specific to each policy head):

```
n_acs = actions.shape[1]
# takes concatenation of image and vector
    encodings as input
fc_out1 = Linear(in_dim=conv_fc.out_dim +
    vect_fc.out_dim, out_dim=32)
fc_out_nl = ReLU()
fc_out2 = Linear(in_dim=fc_out1.out_dim,
    out_dim=n_acs)
```

ψ^{share} (shared across critics for all agents and reward types):

```
state_size = states.shape[1]
fc1 = Linear(in_dim=state_size, out_dim=256)
nl1 = ReLU()
```

$\psi_{i,j}$ (specific to each agent/policy head combination, same architecture for extrinsic and intrinsic critics):

```
n_acs = actions.shape[1]
# fc2 takes other agents' actions as input
fc2 = Linear(in_dim=fc1.out_dim + (num_agents - 1)
    * n_acs, out_dim=256)
nl2 = ReLU()
fc3 = Linear(in_dim=fc2.out_dim, out_dim=n_acs)
```

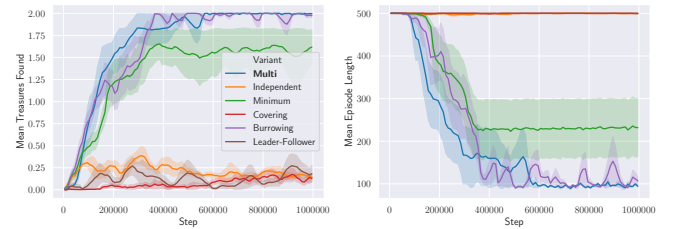## D Training Curves

D.1 Gridworld

Figure 5: Results on Task 1 in Gridworld with 2 agents.

Table 2: Hyperparameter settings across all runs in gridworld.

Name	Description	Value
Q lr	learning rate for centralized critic	0.001
Q optimizer	optimizer for centralized critic	Adam [Kingma and Ba, 2014]
π lr	learning rate for decentralized policies	0.001
π optimizer	optimizer for decentralized policies	Adam
Π lr	learning rate for policy selector	0.04
Π optimizer	optimizer for policy selector	SGD
τ	target function update rate	0.005
bs	batch size	1024
total steps	number of total environment steps	$1e6$
steps per update	number of environment steps between updates	100
niters-model	number of iterations per update for policies and critics	50
niters-selector	number of iterations per update for policy selector	$50^a/2^b$
max ep length	maximum length of an episode before resetting	500
Ψ penalty	coefficient for weight decay on parameters of Q-function	0.001
Θ penalty	coefficient on L_2 penalty on pre-softmax output of policies	0.001
θ penalty	coefficient for weight decay on parameters of policy selector	0.001
$|D|$	maximum size of replay buffer	$1e6$
α	action policy reward scale	100
η	selector policy reward scale	$5^a/0.1^b$
γ	discount factor	0.99
β	relative weight of intrisic rewards to extrinsic	0.1
ζ	decay rate of count-based rewards	0.7

^{a}: **TASKS 1-3**, ^{b}: **FLIP-TASK**
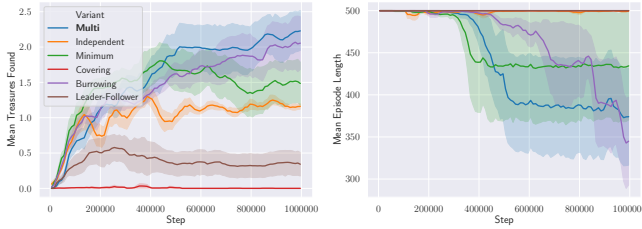
Figure 6: Results on Task 1 in Gridworld with 3 agents.
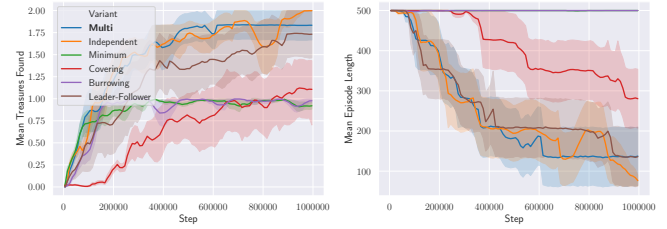
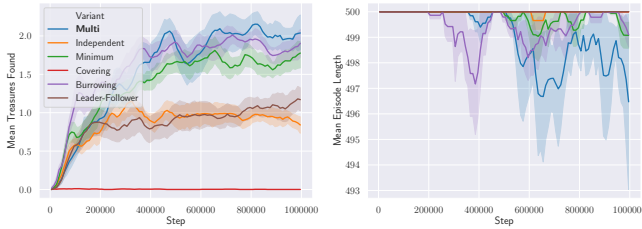Figure 8: Results on Task 2 in Gridworld with 2 agents.

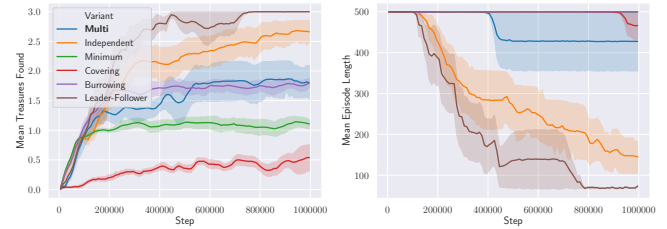Figure 7: Results on Task 1 in Gridworld with 4 agents.

Figure 9: Results on Task 2 in Gridworld with 3 agents.

Table 3: Hyperparameter settings across all runs in VizDoom (only where different from Table 2).

Name	Description	Value
Q lr	learning rate for centralized critic	0.0005
π lr	learning rate for decentralized policies	0.0005
bs	batch size	128
$|D|$	maximum size of replay buffer	$5e5$

Table 4: # of treasures found with standard deviation across 6 runs. Scores are bolded where the best mean score falls within one standard deviation. Our method (**MULTI**) matches or outperforms the non-adaptive oracle in nearly all settings

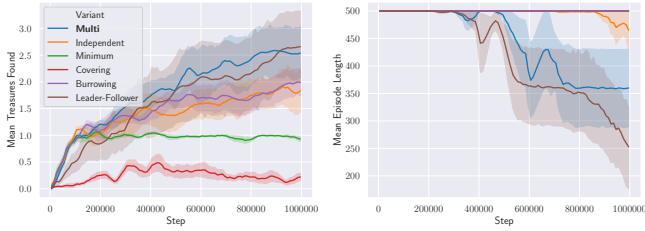|		GRIDWORLD					
		Intrinsic reward type					
Task	n	INDEPENDENT	MINIMUM	COVERING	BURROWING	LEAD-FOLLOW	**MULTI**
1	2	0.14 ± 0.05	$\mathbf{1.62 \pm 0.59}$	0.13 ± 0.12	$\mathbf{1.98 \pm 0.06}$	0.18 ± 0.24	$\mathbf{2.00 \pm 0.00}$
	3	1.16 ± 0.11	$\mathbf{1.49 \pm 0.76}$	0.00 ± 0.00	$\mathbf{2.06 \pm 1.05}$	0.34 ± 0.45	$\mathbf{2.23 \pm 0.73}$
	4	0.84 ± 0.29	$\mathbf{1.78 \pm 0.44}$	0.00 ± 0.00	$\mathbf{1.90 \pm 0.49}$	1.17 ± 0.39	$\mathbf{2.04 \pm 0.61}$
2	2	$\mathbf{2.00 \pm 0.00}$	0.92 ± 0.10	$\mathbf{1.11 \pm 0.99}$	0.98 ± 0.05	$\mathbf{1.73 \pm 0.66}$	$\mathbf{1.83 \pm 0.41}$
	3	$\mathbf{2.66 \pm 0.80}$	1.11 ± 0.29	0.54 ± 0.80	1.80 ± 0.29	$\mathbf{3.00 \pm 0.00}$	1.80 ± 0.71
	4	$\mathbf{1.83 \pm 1.08}$	0.93 ± 0.13	0.22 ± 0.18	$\mathbf{1.99 \pm 0.67}$	$\mathbf{2.66 \pm 2.06}$	$\mathbf{2.54 \pm 1.21}$
3	2	$\mathbf{1.39 \pm 0.94}$	0.67 ± 1.03	0.29 ± 0.37	0.67 ± 1.03	0.83 ± 0.67	$\mathbf{2.00 \pm 0.00}$
	3	$\mathbf{1.68 \pm 0.70}$	0.60 ± 0.73	0.09 ± 0.08	$\mathbf{1.35 \pm 1.16}$	$\mathbf{1.59 \pm 0.83}$	$\mathbf{2.21 \pm 0.91}$
	4	1.12 ± 0.47	1.36 ± 0.71	0.05 ± 0.05	$\mathbf{2.14 \pm 1.49}$	0.68 ± 0.53	$\mathbf{1.73 \pm 0.47}$
Flip-Task	2	$\mathbf{3.04 \pm 1.33}$	$\mathbf{2.66 \pm 1.51}$	2.16 ± 1.47	1.96 ± 1.07	2.73 ± 1.26	$\mathbf{4.03 \pm 0.97}$
		VIZDOOM					
1	2	0.94 ± 0.54	$\mathbf{1.57 \pm 0.74}$	0.16 ± 0.17	$\mathbf{1.94 \pm 0.10}$	0.61 ± 0.43	$\mathbf{1.98 \pm 0.03}$
2		$\mathbf{1.52 \pm 0.75}$	$\mathbf{1.53 \pm 0.74}$	0.70 ± 1.00	0.63 ± 0.04	$\mathbf{1.93 \pm 0.10}$	1.23 ± 0.65
3		0.18 ± 0.19	$\mathbf{0.64 \pm 1.05}$	0.45 ± 0.46	0.29 ± 0.25	0.20 ± 0.17	$\mathbf{1.64 \pm 0.63}$

Figure 10: Results on Task 2 in Gridworld with 4 agents.
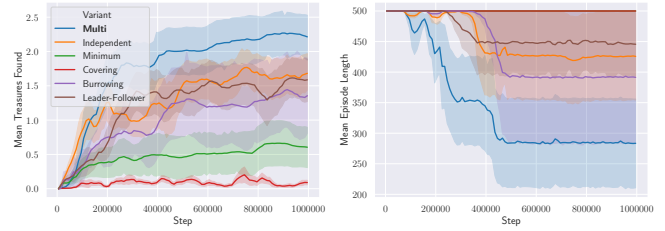

Figure 12: Results on Task 3 in Gridworld with 3 agents.
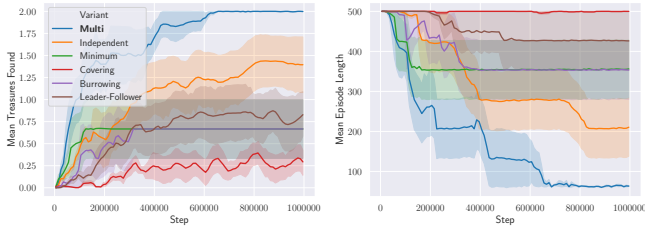

Figure 11: Results on Task 3 in Gridworld with 2 agents.
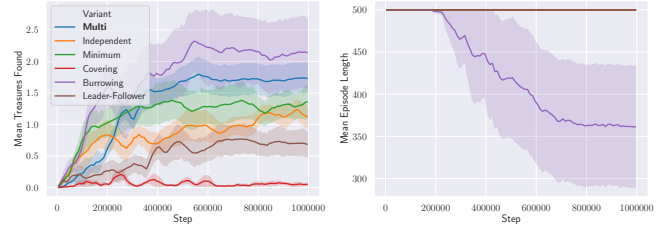

Figure 13: Results on Task 3 in Gridworld with 4 agents.
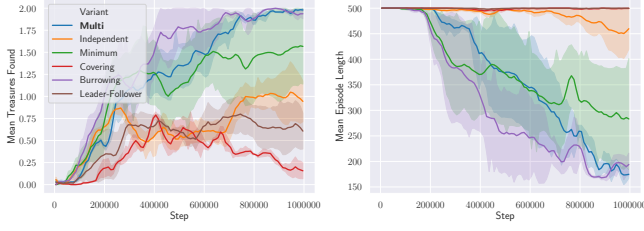
D.2 Vizdoom

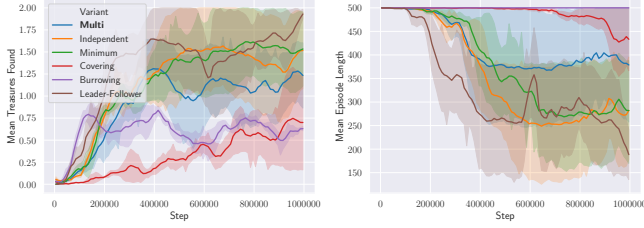Figure 14: Results on Task 1 in Vizdoom.

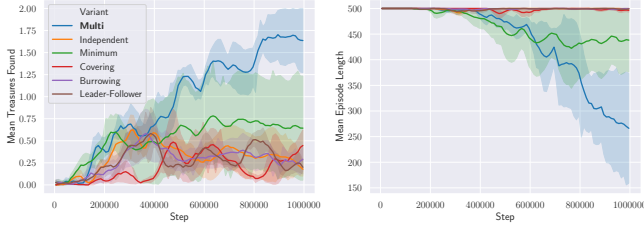
Figure 15: Results on Task 2 in Vizdoom.

Figure 16: Results on Task 3 in Vizdoom.

E Discussion of Multi-Agent Intrinsic Rewards and Their Behaviors

We found the reward types of the non-adaptive oracles (i.e. the reward type that performs best when used exclusively as an intrinsic reward) to largely align with the intuitions behind the design of the tasks. INDEPENDENT rewards, as expected, result in agents exploring the whole state space without taking other agents into consideration. Interestingly, policies trained with these rewards can fail on **TASK 3**, which assigns independent goals to each agent and does not require coordination. Since our maps contain branching non-linear paths, we find that these agents occasionally suffer from detachment [Ecoffet *et al.*, 2019] (ie. forgetting how to return to an previously explored area). Both MINIMUM and BURROWING encourage agents to not explore the same regions redundantly and are thus successful on **TASK 1** which requires divide-and-conquer type behavior. LEADER-FOLLOWER encourages the opposite behavior and is thus most successful on **TASK 2**. COVERING rewards, as expected, lead to behavior where agents are constantly switching the regions they explore. While this behavior alone does not prove to be useful

in the tasks we test since the switching slows down overall exploration progress, it serves as a useful mechanism to break out of bad local optima for our meta-policy approach. The unique behavior of each reward type is visualized in the attached videos.

F More Ablations

In this section we consider more ablations/comparisons to our model across all three tasks in the 2 agent version of gridworld. In the first (*Centralized*) we compute intrinsic rewards under the assumption that all agents are treated as one agent. In other words, we use the inverse count of the number of times that **all** agents have jointly taken up their combined positions, rather than considering agents independently. While this reward function will ensure that the global state space is thoroughly searched, it lacks the inductive biases toward spatial coordination that our reward functions incorporate. As such, it does not learn as efficiently as our method in any of the three tasks. In the second (*Multi (No Entropy)*) we remove the entropy term from the head selector loss function in order to test its importance. We find that this ablation is unable to match the performance of the full method, indicating that entropy is crucial in making sure that our method does not converge early to a suboptimal policy selector. We also compare to the underlying algorithm without intrinsic rewards, multi-agent soft actor-critic (MA-SAC), QMIX [Rashid *et al.*, 2018], and MAVEN [Mahajan *et al.*, 2019]. None of these methods use intrinsic rewards, and we find that they are unable to learn in our setting on any of the tasks, highlighting the significant challenge posed.
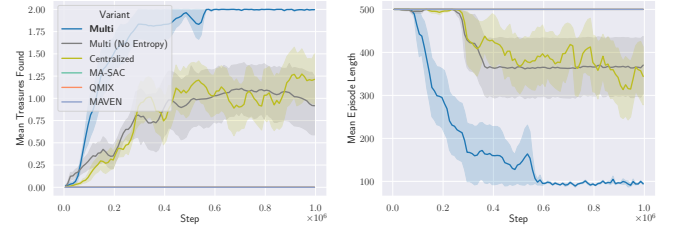

Figure 17: Ablations/Baselines on Task 1 in Gridworld with 2 agents.
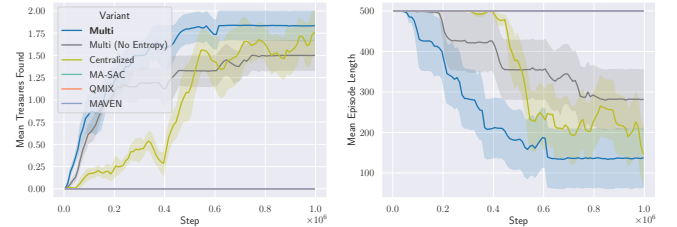

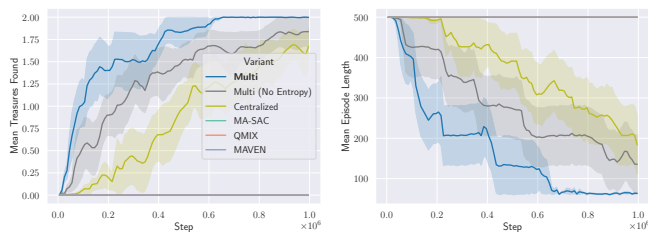Figure 18: Ablations/Baselines on Task 2 in Gridworld with 2 agents.

Figure 19: Ablations/Baselines on Task 3 in Gridworld with 2 agents.

G Analyzing Meta-Policy

In Figure 20 we analyze the behavior of the meta-policy in two separate runs. We evaluate on Task 3, since we find that our method is able to surpass the non-adaptive oracle. This task assigns specific goals to each agent. As such, one might expect that independent exploration would work most effectively in this setting. While independent exploration is effective (see Figure 11), we find that our method can outperform it. In both runs, we find that burrowing rewards are selected when the agents finally learn how to solve the task; however, we find that burrowing rewards are not necessarily successful when deployed on their own. This lack of success is likely due to the fact that these rewards cause the agents to pick a region and commit to exploring it for the duration of training. As such, the agents may pick the "wrong" region at first and never be able to recover. On the other hand, using our methods, the meta-policy can wait until the burrowing exploration regions align with the assigned rewards and then select the policies trained on these rewards. This usually ends up being more efficient than waiting for the agents to explore the whole map using independent rewards and can minimize the likelihood of detachment [Ecoffet *et al.*, 2019].
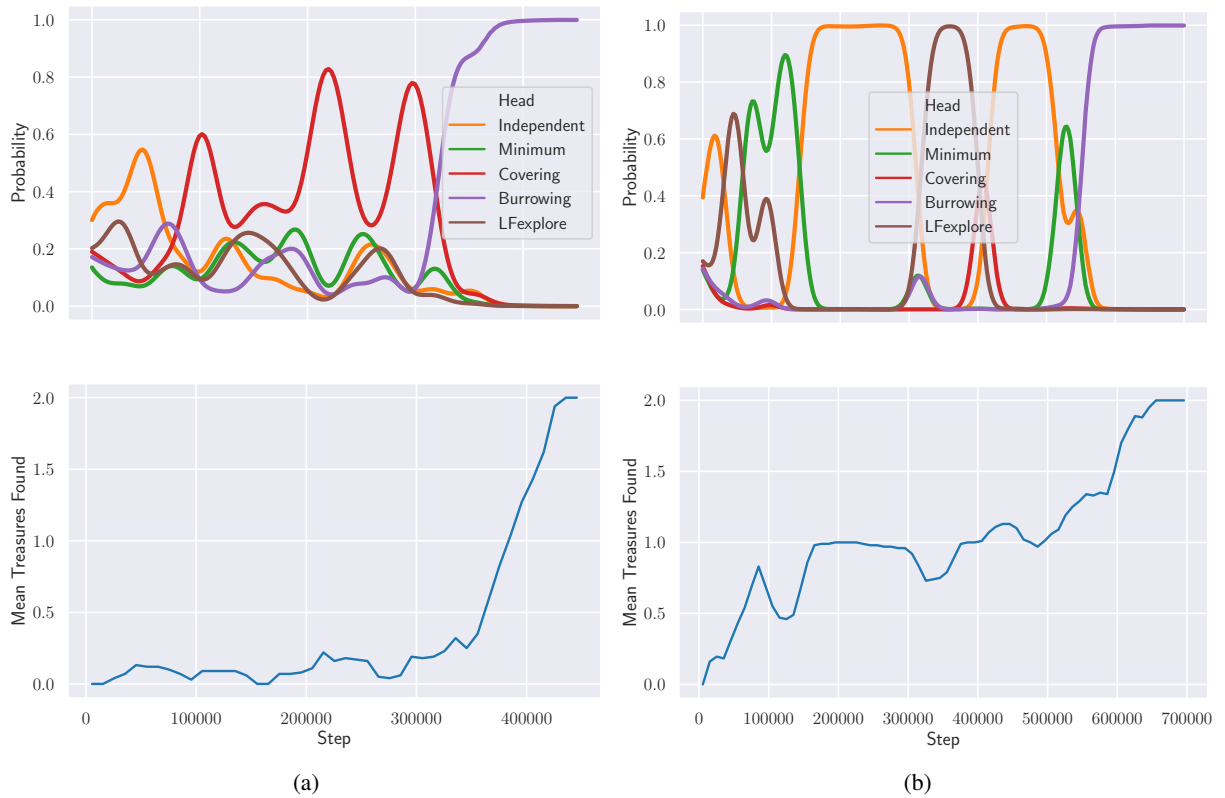
Figure 20: Two runs of our method on GridWorld Task 3 with 2 agents. Top row shows the evolution of the meta-policy's probability of selecting each policy head. Bottom row shows the number of treasures found per episode.