Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards

Alexander Trott
Salesforce Research
atrott@salesforce.com

Stephan Zheng
Salesforce Research
stephan.zheng@salesforce.com

Caiming Xiong
Salesforce Research
cxiong@salesforce.com

Richard Socher
Salesforce Research
rsocher@salesforce.com

Abstract

While using shaped rewards can be beneficial when solving sparse reward tasks, their successful application often requires careful engineering and is problem specific. For instance, in tasks where the agent must achieve some goal state, simple distance-to-goal reward shaping often fails, as it renders learning vulnerable to local optima. We introduce a simple and effective model-free method to learn from shaped distance-to-goal rewards on tasks where success depends on reaching a goal state. Our method introduces an auxiliary distance-based reward based on *pairs* of rollouts to encourage diverse exploration. This approach effectively prevents learning dynamics from stabilizing around local optima induced by the naive distance-to-goal reward shaping and enables policies to efficiently solve sparse reward tasks. Our augmented objective does not require any additional reward engineering or domain expertise to implement and converges to the original sparse objective as the agent learns to solve the task. We demonstrate that our method successfully solves a variety of hard-exploration tasks (including maze navigation and 3D construction in a Minecraft environment), where naive distance-based reward shaping otherwise fails, and intrinsic curiosity and reward relabeling strategies exhibit poor performance.

1 Introduction

Reinforcement Learning (RL) offers a powerful framework for teaching an agent to perform tasks using only observations from its environment. Formally, the goal of RL is to learn a policy that will maximize the reward received by the agent; for many real-world problems, this requires access to or engineering a reward function that aligns with the task at hand. Designing a well-suited *sparse* reward function simply requires defining the criteria for solving the task: reward is provided if the criteria for completion are met and withheld otherwise. While designing a suitable sparse reward may be straightforward, learning from it within a practical amount of time often is not, often requiring exploration heuristics to help an agent discover the sparse reward (Pathak et al., 2017; Burda et al., 2018b,a). *Reward shaping* (Mataric, 1994; Ng et al., 1999) is a technique to modify the reward signal, and, for instance, can be used to relabel and learn from failed rollouts, based on which ones made more progress towards task completion. This may simplify some aspects of learning, but whether the learned behavior improves task performance depends critically on careful design of the shaped reward (Clark & Amodei, 2016). As such, reward shaping requires domain-expertise and is often problem-specific (Mataric, 1994).

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Tasks with well-defined goals provide an interesting extension of the traditional RL framework (Kaelbling, 1993; Sutton et al., 2011; Schaul et al., 2015). Such tasks often require RL agents to deal with goals that vary across episodes and define success as achieving a state within some distance of the episode's goal. Such a setting naturally defines a sparse reward that the agent receives when it achieves the goal. Intuitively, the same distance-to-goal measurement can be further used for reward shaping (without requiring additional domain-expertise), given that it measures progress towards success during an episode. However, reward shaping often introduces new local optima that can prevent agents from learning the optimal behavior for the original task. In particular, the existence and distribution of local optima strongly depends on the environment and task definition.

As such, successfully implementing reward shaping quickly becomes problem specific. These limitations have motivated the recent development of methods to enable learning from sparse rewards (Schulman et al., 2017; Liu et al., 2019), methods to learn latent representations that facilitate shaped reward (Ghosh et al., 2018; Nair et al., 2018; Warde-Farley et al., 2019), and learning objectives that encourage diverse behaviors (Haarnoja et al., 2017; Eysenbach et al., 2019).

We propose a simple and effective method to address the limitations of using distance-to-goal as a shaped reward. In particular, we extend the naive distance-based shaped reward to handle *sibling* trajectories, pairs of independently sampled trajectories using the same policy, starting state, and goal. Our approach, which is simple to implement, can be interpreted as a type of self-balancing reward: we encourage behaviors that make progress towards the goal and simultaneously use sibling rollouts to estimate the local optima and encourage behaviors that avoid these regions, effectively balancing exploration and exploitation. This objective helps to *de*-stabilize local optima without introducing new stable optima, preserving the task definition given by the sparse reward. This additional objective also relates to the entropy of the distribution of terminal states induced by the policy; however, unlike other methods to encourage exploration (Haarnoja et al., 2017), our method is "self-scheduling" such that our proposed shaped reward converges to the sparse reward as the policy learns to reach the goal.

Our method combines the learnability of shaped rewards with the generality of sparse rewards, which we demonstrate through its successful application on a variety of environments that support goal-oriented tasks. In summary, our contributions are as follows:

- We propose Sibling Rivalry, a method for model-free, dynamic reward shaping that preserves optimal policies on sparse-reward tasks.
- We empirically show that Sibling Rivalry enables RL agents to solve hard-exploration sparse-reward tasks, where baselines often struggle to learn. We validate in four settings, including continuous navigation and discrete bit flipping tasks as well as hierarchical control for 3D navigation and 3D construction in a demanding Minecraft environment.

2 Preliminaries

Consider an agent that must learn to maximize some task reward through its interactions with its environment. At each time point t throughout an episode, the agent observes its state $s_t \in S$ and selects an action $a_t \in A$ based on its policy $\pi(a_t|s_t)$, yielding a new state s'_t sampled according to the environment's transition dynamics $p(s'_t|s_t, a_t)$ and an associated reward r_t governed by the task-specific reward function $r(s_t, a_t, s'_t)$. Let $\boldsymbol{\tau} = \{(s_t, a_t, s'_t, r_t)\}_{t=0}^{T-1}$ denote the trajectory of states, actions, next states, and rewards collected during an episode of length T, where T is determined by either the maximum episode length or some task-specific termination conditions. The objective of the agent is to learn a policy that maximizes its expected cumulative reward: $\mathbb{E}_{\boldsymbol{\tau}\sim\pi,p}\left[\Sigma_t \gamma^t r_t\right]$.

Reinforcement Learning for Goal-oriented tasks. The basic RL framework can be extended to a more general setting where the underlying association between states, actions, and reward can change depending on the parameters of a given episode (Sutton et al., 2011). From this perspective, the agent must learn to optimize a *set* of potential rewards, exploiting the shared structure of the individual tasks they each represent. This is applicable to the case of learning a *goal-conditioned* policy $\pi(a_t|s_t, g)$. Such a policy must embed a sufficiently generic understanding of its environment to choose whatever actions lead to a state consistent with the goal g (Schaul et al., 2015). This setting naturally occurs whenever a task is defined by some set of goals G that an agent must learn to reach when instructed. Typically, each episode is structured around a specific goal $g \in G$ sampled from the

task distribution. In this work, we make the following assumptions in our definition of "goal-oriented task":

1. The task defines a distribution over starting states and goals $\rho(s_0, g)$ that are sampled to start each episode.
2. Goals can be expressed in terms of states such that there exists a function $m(s) : S \to G$ that maps state s to its equivalent goal.
3. $S \times G \to \mathbb{R}^+$ An episode is considered a success once the state is within some radius of the goal, such that $d(s, g) \leq \delta$, where $d(x, y) : G \times G \to \mathbb{R}^+$ is a distance function[1] and $\delta \in \mathbb{R}^+$ is the distance threshold. (Note: this definition is meant to imply that the distance function internally applies the mapping m to any states that are used as input; we omit this from the notation for brevity.)

This generic task definition allows for an equally generic sparse reward function $r(s, g)$:

$$r(s, g) = \begin{cases} 1, & d(s, g) \leq \delta \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

From this, we define $r_t \triangleq r(s'_t, g)$ so that reward at time t depends on the state reached after taking action a_t from state s_t. Let us assume for simplicity that an episode terminates when either the goal is reached or a maximum number of actions are taken. This allows us to define a single reward for an entire trajectory considering only the terminal state, giving: $r_{\boldsymbol{\tau}} \triangleq r(s_T, g)$, where s_T is the state of the environment when the episode terminates. The learning objective now becomes finding a goal-conditioned policy that maximizes $\mathbb{E}_{\boldsymbol{\tau} \sim \pi, p,\ s_0, g \sim \rho}\left[r_{\boldsymbol{\tau}}\right]$.

3 Approach

Distance-based shaped rewards and local optima. We begin with the observation that the distance function d (used to define goal completion and compute sparse reward) may be exposed as a shaped reward without any additional domain knowledge:

$$\tilde{r}(s, g) = \begin{cases} 1, & d(s, g) \leq \delta \\ -d(s, g), & \text{otherwise} \end{cases}, \qquad \tilde{r}_{\boldsymbol{\tau}} \triangleq \tilde{r}(s_T, g). \tag{2}$$

By definition, a state that globally optimizes $\tilde{r}$ also achieves the goal (and yields sparse reward), meaning that $\tilde{r}$ preserves the global optimum of r. While we expect the distance function itself to have a single (global) optimum with respect to s and a fixed g, in practice we need to consider the possibility that other *local* optima exist because of the state space structure, transition dynamics and other features of the environment. For example, the agent may need to *increase* its distance to the goal in order to eventually reach it. This is exactly the condition faced in the toy task depicted in Figure 1. We would like to gain some intuition for how the learning dynamics are influenced by such local optima and how this influence can be mitigated.

The "learning dynamics" refer to the interaction between (i) the distribution of terminal states $\rho_g^\pi(s_T)$ induced by a policy π in pursuit of goal g and (ii) the optimization of the policy with respect to $\mathbb{E}_{\rho_g^\pi(s_T)}[\tilde{r}(s_T, g)]$. A local optimum $o_g \in S$ can be considered "stable" if, for all policies within some basin of attraction, continued optimization causes $\rho_g^\pi(s_T)$ to converge to o_g. Figure 1 (middle) presents an example of this. The agent observes its 2D position along the track and takes an action to change its position; its reward is based on its terminal state (after 5 steps). Because of its starting position, maximizing the naive reward $\tilde{r}(s, g)$ causes the policy to "get stuck" at the local optimum o_g, i.e., the final state $\rho_g^\pi(s_T)$ is peaked around o_g.

In this example, the location of the local optimum is obvious and we can easily engineer a reward bonus for avoiding it. In its more general form, this augmented reward is:

$$r'(s, g, \bar{g}) = \begin{cases} 1, & d(s, g) \leq \delta \\ \min\left[0, -d(s, g) + d(s, \bar{g})\right], & \text{otherwise} \end{cases}, \qquad r'_{\boldsymbol{\tau}} \triangleq r'(s_T, g, \bar{g}). \tag{3}$$

[1] A straightforward metric, such as L_1 or L_2 distance, is often sufficient to express goal completion.

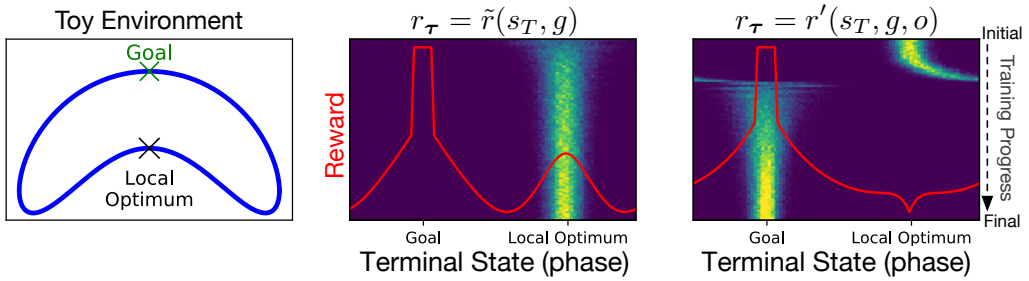

Figure 1: **Motivating example.** (Left) The agent's task is to reach the goal (green X) by controlling its position along a warped circular track. A distance-to-goal reward (L_2 distance) creates a local optimum o_g (black X). (Middle and Right) Terminal state distributions during learning. The middle figure shows optimization using a distance-to-goal shaped reward. For the right figure, the shaped reward is augmented to include a hand-engineered bonus for avoiding o_g (Eq. 3; $\bar{g} \leftarrow o_g$). The red overlay illustrates the reward at each phase of the track.

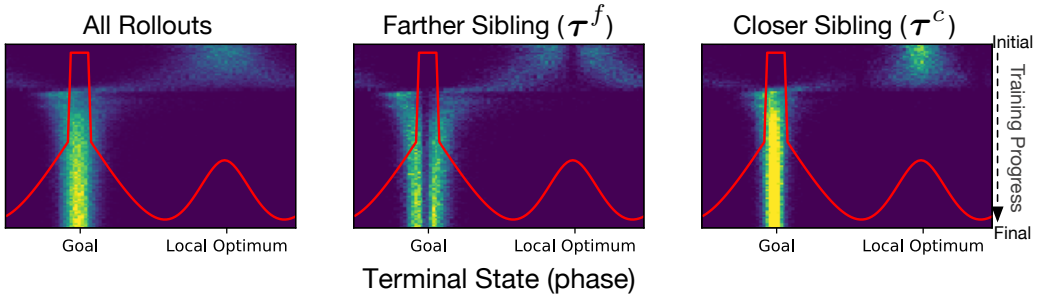

Figure 2: **Learning with Sibling Rivalry.** Terminal state distribution over training when using SR. Middle and right plots show the farther $\boldsymbol{\tau}^f$ and closer $\boldsymbol{\tau}^c$ trajectories, respectively. Red overlay illustrates the shape of the naive distance-to-goal reward $\tilde{r}$.

where $\bar{g} \in G$ acts as an 'anti-goal' and specifies a state that the agent should avoid, e.g., the local optimum o_g in the case of the toy task in Figure 1. Indeed, using r' and setting $\bar{g} \leftarrow o_g$ (that is, using o_g as the 'anti-goal'), prevents the policy from getting stuck at the local optimum and enables the agent to quickly learn to reach the goal location (Figure 1, right).

While this works well in this toy setting, the intuition for which state(s) should be used as the 'anti-goal' $\bar{g}$ will vary depending on the environment, the goal g, and learning algorithm. In addition, using a fixed $\bar{g}$ may be self-defeating if the resulting shaped reward introduces its own new local optima. To make use of $r'(s, g, \bar{g})$ in practice, we require a method to dynamically estimate the local optima that frustrate learning without relying on domain-expertise or hand-picked estimations.

Self-balancing reward. We propose to estimate local optima directly from the behavior of the policy by using *sibling* rollouts. We define a pair of sibling rollouts as two independently sampled trajectories sharing the same starting state s_0 and goal g. We use the notation $\boldsymbol{\tau}^f, \boldsymbol{\tau}^c \sim \pi | g$ to denote a pair of trajectories from 2 sibling rollouts, where the superscript specifies that $\boldsymbol{\tau}^c$ ended closer to the goal than $\boldsymbol{\tau}^f$, i.e. that $\tilde{r}_{\boldsymbol{\tau}^c} \geq \tilde{r}_{\boldsymbol{\tau}^f}$. By definition, optimization should tend to bring $\boldsymbol{\tau}^f$ closer towards $\boldsymbol{\tau}^c$ during learning. That is, it should make $\boldsymbol{\tau}^f$ less likely and $\boldsymbol{\tau}^c$ more likely. In other words, the terminal state of the closer rollout s_T^c can be used to estimate the location of local optima created by the distance-to-goal shaped reward.

To demonstrate this, we revisit the toy example presented in Figure 1 but introduce paired sampling to produce sibling rollouts (Figure 2). As before, we optimize the policy using r' but with 2 important modifications. First, we use the sibling rollouts for *mutual relabeling* using the augmented shaped

Algorithm 1: Sibling Rivalry

Given

- Environment, Goal-reaching task w/ $S, G, A, \rho(s_0, g), m(), d(,), \delta$ and max episode length
- Policy $\pi : S \times G \times A \to [0, 1]$ and Critic $V : S \times G \times G \to \mathbb{R}$ with parameters θ
- On-policy learning algorithm $\mathbb{A}$, e.g., REINFORCE, Actor-critic, PPO
- Inclusion threshold ϵ

for *iteration = 1...K* **do**
  Initialize transition buffer D
  for *episode = 1...M* **do**
    Sample $s_0, g \sim \rho$
    $\boldsymbol{\tau}^a \leftarrow \pi_\theta(...)|_{s_0,g}$ # Collect rollout
    $\boldsymbol{\tau}^b \leftarrow \pi_\theta(...)|_{s_0,g}$ # Collect sibling rollout
    Relabel $\boldsymbol{\tau}^a$ reward using r' and $\bar{g} \leftarrow m(s_T^b)$
    Relabel $\boldsymbol{\tau}^b$ reward using r' and $\bar{g} \leftarrow m(s_T^a)$
    if $d(s_T^a, g) < d(s_T^b, g)$ **then**
      $\boldsymbol{\tau}^c \leftarrow \boldsymbol{\tau}^a$
      $\boldsymbol{\tau}^f \leftarrow \boldsymbol{\tau}^b$
    else
      $\boldsymbol{\tau}^c \leftarrow \boldsymbol{\tau}^b$
      $\boldsymbol{\tau}^f \leftarrow \boldsymbol{\tau}^a$
    if $d(s_T^c, s_T^f) < \epsilon$ ***or*** $d(s_T^c, g) < \delta$ **then**
      Add $\boldsymbol{\tau}^f$ and $\boldsymbol{\tau}^c$ to buffer D
    else
      Add $\boldsymbol{\tau}^f$ to buffer D
  Apply on-policy algorithm $\mathbb{A}$ to update θ using examples in D

reward r' (Eq. 3), where each rollout treats its sibling's terminal state as its own anti-goal:

$$r'_{\boldsymbol{\tau}^f} = r'(s_T^f, g, s_T^c) \quad \& \quad r'_{\boldsymbol{\tau}^c} = r'(s_T^c, g, s_T^f). \tag{4}$$

Second, we only include the closer-to-goal trajectory $\boldsymbol{\tau}^c$ for computing policy updates if it reached the goal. As shown in the distribution of s_T^c over training (Figure 2, right), s_T^c remains closely concentrated around *an* optimum: the local optimum early in training and later the global optimum g. Our use of sibling rollouts creates a reward signal that intrinsically balances exploitation and exploration by encouraging the policy to minimize distance-to-goal while de-stabilizing local optima created by that objective. Importantly, as the policy converges towards the *global* optimum (i.e. learns to reach the goal), r' converges to the original underlying sparse reward r.

Sibling Rivalry. From this, we derive a more general method for learning from sibling rollouts: Sibling Rivalry (SR). Algorithm 1 describes the procedure for integrating SR into existing on-policy algorithms for learning in the settings we described above. SR has several key features:

1. sampling sibling rollouts,
2. mutual reward relabeling based on our self-balancing reward r',
3. selective exclusion of $\boldsymbol{\tau}^c$ (the closer rollout) trajectories from gradient estimation, using hyperparameter $\epsilon \in \mathbb{R}^+$ for controlling the inclusion/exclusion criterion.

Consistent with the intuition presented above, we find that ignoring $\boldsymbol{\tau}^c$ during gradient estimation helps prevent the policy from converging to local optima. In practice, however, it can be beneficial to learn directly from $\boldsymbol{\tau}^c$. The hyperparameter ϵ serves as an inclusion threshold for controlling when $\boldsymbol{\tau}^c$ is included in gradient estimation, such that SR always uses $\boldsymbol{\tau}^f$ for gradient estimation and includes $\boldsymbol{\tau}^c$ only if it reaches the goal or if $d(s_T^f, s_T^c) \leq \epsilon$.

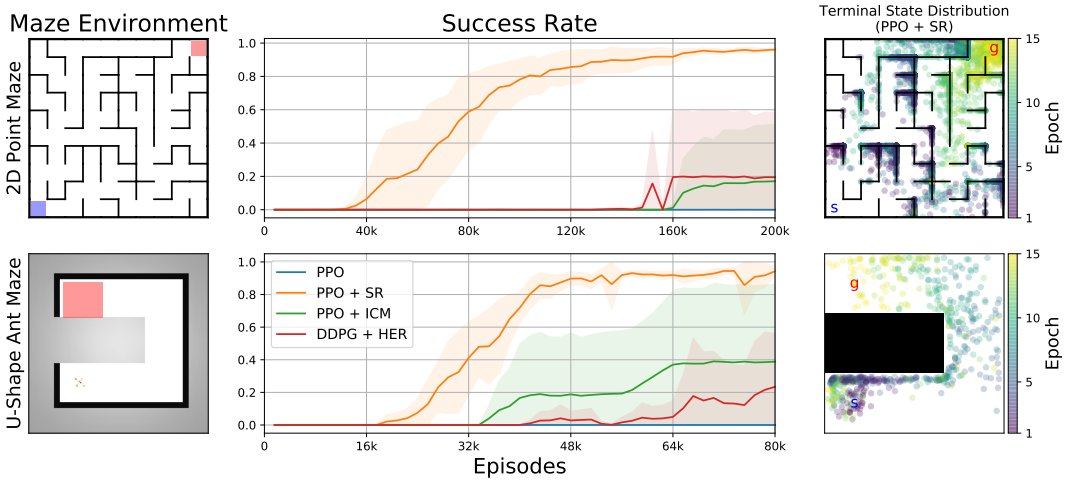

Figure 3: **(Left) Maze environments.** Top row illustrates our 2D point maze; bottom row shows the U-shaped Ant Maze in a Mujoco environment. For the 2D maze, start location is sampled within the blue square; in the ant maze, the agent starts near its pictured location. For both, the goal is randomly sampled from within the red square region. **(Middle) Learning progress.** Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean±SD, clipped to [0, 1]). Only our method (PPO+SR) allows the agent to discover the goal in all experiments. Conversely, PPO with the naive distance-to-goal reward never succeeds. Methods to learn from sparse rewards (PPO+ICM and DDPG+HER) only rarely discover the goals. Episodes have a maximum duration of 50 and 500 environment steps for the 2D Point Maze and Ant Maze, respectively. **(Right) State distribution.** Colored points illustrate terminal states achieved by the policy after each of the first 15 evaluation checkpoints. PPO+SR allows the agent to discover increasingly good optima without becoming stuck in them.

The toy example above (Figure 2) shows an instance of using SR where the base algorithm is A2C, the environment only yields end-of-episode reward ($\gamma = 1$), and the closer rollout $\boldsymbol{\tau}^c$ is only used in gradient estimation when that rollout reaches the goal ($\epsilon = 0$). In our below experiments we mostly use end-of-episode rewards, although SR does not place any restriction on this choice. It should be noted, however, that our method does require that full-episode rollouts are sampled in between parameter updates (based on the choice of treating the *terminal* state of the sibling rollout as $\bar{g}$) and that experimental control over episode conditions (s_0 and g) is available.[2] Lastly, we point out that we include the state s_t, episode goal g, and anti-goal $\bar{g}$ as inputs to the critic network V; the policy π sees only s_t and g.

In the appendix, we present a more formal motivation of the technique (Section A), additional clarifying examples addressing the behavior of SR at different degrees of local optimum severity (Section B), and an empirical demonstration (Section C) showing how ϵ can be used to tune the system towards exploration ($\downarrow \epsilon$) or exploitation ($\uparrow \epsilon$).

4 Experiments

To demonstrate the effectiveness of our method, we apply it to a variety of goal-reaching tasks. We focus on settings where local optima interfere with learning from naive distance-to-goal shaped rewards. We compare this baseline to results using our approach as well as to results using curiosity and reward-relabeling in order to learn from sparse rewards. The appendix (Section F) provides detailed descriptions of the environments, tasks, and implementation choices.

2D Point-Maze Navigation. How do different training methods handle the exploration challenge that arises in the presence of numerous local optima? To answer this, we train an agent to navigate a fully-continuous 2D point-maze with the configuration illustrated in Figure 3 (top left). At each point

[2]Though we observe SR to work when s_0 is allowed to differ between sibling rollouts (appendix, Sec. D)

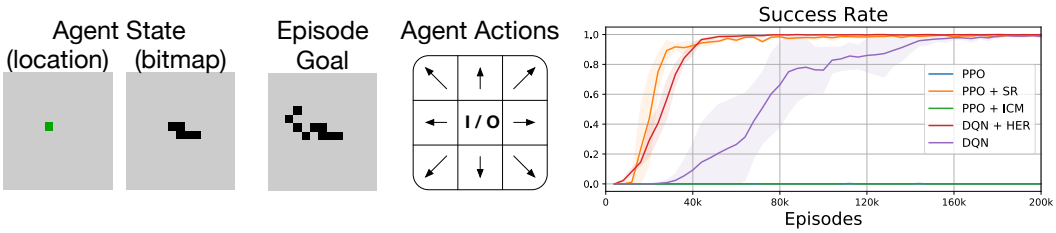

Figure 4: **2D discrete pixel-grid environment.** The agent begins in a random location on a 13x13 grid with all pixels off and must move and toggle pixels to produce the goal bitmap. The agent sees its current location (1-hot), the current bitmap, and the goal bitmap. The agent succeeds when the bitmap exactly matches the goal (0-distance). Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean±SD, clipped to [0, 1]). Episodes have a maximum duration of 50 environment steps.

in time, the agent only receives its current coordinates and the goal coordinates. It outputs an action that controls its change in location; the actual change is affected by collisions with walls. When training using Proximal Policy Optimization (Schulman et al., 2017) and a shaped distance-to-goal reward, the agent consistently learns to exploit the corridor at the top of the maze but never reaches the goal. Through incorporating Sibling Rivalry (PPO + SR), the agent avoids this optimum (and all others) and discovers the path to the goal location, solving the maze.

We also examine the behavior of algorithms designed to enable learning from sparse rewards without reward shaping. Hindsight Experience Replay (HER) applies off-policy learning to relabel trajectories based on achieved goals (Andrychowicz et al., 2017). In this setting, HER [using a DDPG backbone (Lillicrap et al., 2016)] only learns to reach the goal on 1 of the 5 experimental runs, suggesting a failure in exploration since the achieved goals do not generalize to the task goals. Curiosity-based intrinsic reward (ICM), which is shown to maintain a curriculum of exploration (Pathak et al., 2017; Burda et al., 2018a), fails to discover the sparse reward at the same rate. Using random network distillation (Burda et al., 2018b), a related intrinsic motivation method, the agent never finds the goal (not shown for visual clarity). Only the agent that learns with SR is able to consistently and efficiently solve the maze (Figure 3, top middle).

Ant-Maze Navigation using Hierarchical RL. SR easily integrates with HRL, which can help to solve more difficult problems such as navigation in a complex control environment (Nachum et al., 2018). We use HRL to solve a U-Maze task with a Mujoco (Todorov et al., 2012) ant agent (Figure 3, bottom left), requiring a higher-level policy to propose subgoals based on the current state and the goal of the episode as well as a low-level policy to control the ant agent towards the given subgoal. For fair comparison, we employ a standardized approach for training the low-level controller from subgoals using PPO but vary the approach for training the high-level controller. For this experiment, we restrict the start and goal locations to the opposite ends of the maze (Figure 3, bottom left).

The results when learning to navigate the ant maze corroborate those in the toy environment: learning from the naive distance-to-goal shaped reward $\tilde{r}$ fails because the wall creates a local optimum that policy gradient is unable to escape (PPO). As with the 2D Point Maze, SR can exploit the optimum without becoming stuck in it (PPO+SR). This is clearly visible in the terminal state patterns over early training (Figure 3, bottom right). We again compare with methods to learn from sparse rewards, namely HER and ICM. As before, ICM stochastically discovers a path to the goal but at a low rate (2 in 5 experiments). In this setting, HER struggles to generalize from its achieved goals to the task goals, perhaps due in part to the difficulties of off-policy HRL (Nachum et al., 2018). 3 of the 5 HER runs eventually discover the goal but do not reach a high level of performance.

Application to a Discrete Environment. Distance-based rewards can create local optima in less obvious settings as well. To examine such a setting and to show that our method can apply to environments with discrete action/state spaces, we experiment with learning to manipulate a 2D bitmap to produce a goal configuration. The agent starts in a random location on a 13x13 grid and may move to an adjacent location or toggle the state of its current position (Figure 4, left). We use L_1 distance (that is, the sum of bitwise absolute differences). Interestingly, this task does not require the

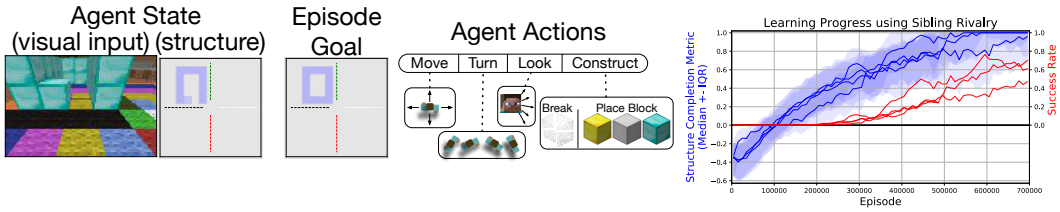

Figure 5: **3D construction task in Minecraft.** The agent must control its location/orientation and break/place blocks in order to produce the goal structure. The agent observes its first-person visual input, the discrete 3D cuboid of the construction arena, and the corresponding cuboid of the goal structure. An episode is counted as a success when the structure exactly matches the goal. The *Structure Completion Metric* is difference between correctly and incorrectly placed blocks divided by the number of goal-structure blocks. In the illustrated example, the agent has nearly constructed the goal, which specifies a height-2 diamond structure near the top left of the construction arena. Goal structures vary in height, dimensions, and material (4806 unique combinations). Episodes have a maximum duration of 100 environment steps.

agent to increase the distance to the goal in order to reach it (as, for example, with the Ant Maze), but naive distance-to-goal reward shaping still creates 'local optima' by introducing pathological learning dynamics: early in training, when behavior is closer to random, toggling a bit from off to on tends to *increase* distance-to-goal and the agent quickly learns to avoid taking the toggle action. Indeed, the agents trained with naive distance-to-goal reward shaping $\tilde{r}$ never make progress (PPO). As shown in Figure 4, we can prevent this outcome and allow the agent to learn the task through incorporating Sibling Rivalry (PPO+SR).

As one might expect, off-policy methods that can accommodate forced exploration may avoid this issue; DQN (Mnih et al., 2015) gradually learns the task (note: this required densifying the reward rather than using only the terminal state). However, exploration alone is not sufficient on a task like this since simply achieving diverse states is unlikely to let the agent discover the task structure relating states, goals, and rewards, as evidenced by the failure of ICM to enable learning in this setting. HER aims to learn this task structure from failed rollouts and, as an off-policy method, handles forced exploration, allowing it to quickly learn this task. Intuitively, using distance as a reward signal automatically exposes the task structure but often at the cost of unwanted local optima. Sibling Rivalry avoids that tradeoff, allowing efficient on-policy learning[3].

3D Construction in Minecraft. Finally, to demonstrate that Sibling Rivalry can be applied to learning in complex environments, we apply it to a custom 3D construction task in Minecraft using the Malmo platform (Johnson et al., 2016). Owing to practical limitations, we use this setting to illustrate the scalability of SR rather than to provide a detailed comparison with other methods. Similar to the pixel-grid task, here the agent must produce a discrete goal structure by placing and removing blocks (Figure 5). However, this task introduces the challenge of a first-person 3D environment, combining continuous and discrete inputs, and application of aggressively asynchronous training with distributed environments [making use of the IMPALA framework (Espeholt et al., 2018)]. Since success requires exact-match between the goal and constructed cuboids, we use the number of block-wise differences as our distance metric. Using this distance metric as a naive shaped reward causes the agent to avoid ever placing blocks within roughly 1000 episodes (not shown for visual clarity). Simply by incorporating Sibling Rivalry the agent avoids this local optimum and learns to achieve a high degree of construction accuracy and rate of exact-match success (Figure 5, right).

5 Related Work

Intrinsic Motivation. Generally speaking, the difficulty in learning from sparse rewards comes from the fact that they tend to provide prohibitively rare signal to a randomly initialized agent. Intrinsic motivation describes a form of task-agnostic reward shaping that encourages exploration by rewarding novel states. Count-based methods track how often each state is visited to reward

[3]We find that including both sibling trajectories ($\epsilon = \texttt{Inf}$) works best in the discrete-distance settings

reaching relatively unseen states (Bellemare et al., 2016; Tang et al., 2017). Curiosity-driven methods encourage actions that surprise a separate model of the network dynamics (Pathak et al., 2017; Burda et al., 2018a; Zhao & Tresp, 2018). Burda et al. (2018b) introduce a similar technique using distillation of a random network. In addition to being more likely to discover sparse reward, policies that produce diverse coverage of states provide a strong initialization for downstream tasks (Haarnoja et al., 2017; Eysenbach et al., 2019). Intrinsic motivation requires that the statistics of the agent's experience be directly tracked or captured in the training progress of some external module. In contrast, we use the policy itself to estimate and encourage exploratory behavior.

Curriculum Learning and Self-Play. Concepts from curriculum learning (Bengio et al., 2009) have been applied to facilitate learning goal-directed tasks (Molchanov et al., 2018; Nair et al., 2018). Florensa et al. (2018), for example, introduce a generative adversarial network approach for automatic generation of a goal curriculum. On competitive tasks, such as 2-player games, self-play has enabled remarkable success (Silver et al., 2018). Game dynamics yield balanced reward and force agents to avoid over-committing to suboptimal strategies, providing both a natural curriculum and incentive for exploration. Similar benefits have been gained through asymmetric self-play with goal-directed tasks (Sukhbaatar et al., 2018a,b). Our approach shares some inspiration with this line of work but combines the asymmetric objectives into a single reward function.

Learning via Generalization. Hindsight Experience Replay (Andrychowicz et al., 2017) combines reward relabeling and off-policy methods to allow learning from sparse reward even on failed rollouts, leveraging the generalization ability of neural networks as universal value approximators (Schaul et al., 2015). Asymmetric competition has been used to improve this method, presumably by inducing an automatic exploration curriculum that helps relieve the generalization burden (Liu et al., 2019).

Latent Reward Shaping. A separate approach within reward shaping involves using latent representations of goals and states. Ghosh et al. (2018) estimate distance between two states based on the actions a pre-trained policy would take to reach them. Nair et al. (2018) introduce a method for unsupervised learning of goal spaces that allows practicing reaching imagined goal states by computing distance in latent space [see also Péré et al. (2018)]. Warde-Farley et al. (2019) use discriminitive training to learn to estimate similarity to a goal state from raw observations.

6 Conclusion

We introduce Sibling Rivalry, a simple and effective method for learning goal-reaching tasks from a generic class of distance-based shaped rewards. Sibling Rivalry makes use of sibling rollouts and self-balancing rewards to prevent the learning dynamics from stabilizing around local optima. By leveraging the distance metric used to define the underlying sparse reward, our technique enables robust learning from shaped rewards without relying on carefully-designed, problem-specific reward functions. We demonstrate the applicability of our method across a variety of goal-reaching tasks where naive distance-to-goal reward shaping consistently fails and techniques to learn from sparse rewards struggle to explore properly and/or generalize from failed rollouts. Our experiments show that Sibling Rivalry can be readily applied to both continuous and discrete domains, incorporated into hierarchical RL, and scaled to demanding environments.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *NIPS*, 2017.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *NIPS*, 2016.

Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *ICML*, 2009.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-Scale Study of Curiosity-Driven Learning. *arXiv*, 2018a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation. *arXiv*, 2018b.

Po-Wei Chou, Maturana Daniel, and Sebastian Scherer. Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution. In *ICML*, 2017.

Jack Clark and Dario Amodei. Faulty reward functions in the wild. `https://openai.com/blog/faulty-reward-functions/`, 2016.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *ICML*, 2018.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity Is All You Need: Learning Skills Without a Reward Function. In *ICLR*, 2019.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic Goal Generation for Reinforcement Learning Agents. In *ICML*, 2018.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning Actionable Representations with Goal-Conditioned Policies. *arXiv*, 2018.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies. In *ICML*, 2017.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for Artificial Intelligence Experimentation. *IJCAI*, 2016.

Leslie Pack Kaelbling. Learning to Achieve Goals. In *IJCAI*, 1993.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Hao Liu, Alexander Trott, Richard Socher, and Caiming Xiong. Competitive Experience Replay. In *ICLR*, 2019.

Maja J Mataric. Reward Functions for Accelerated Learning. In *ICML*, 1994.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33, 2015. ISSN 1476-4687.

Artem Molchanov, Karol Hausman, Stan Birchfield, and Gaurav Sukhatme. Region Growing Curriculum Generation for Reinforcement Learning. *arXiv*, 2018.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. *arXiv*, 2018.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual Reinforcement Learning with Imagined Goals. *arXiv*, 2018.

Andrew Y Ng, Daishi Harada, and Stuart Russel. Policy invariance under reward transformations: theory and application to reward shaping. In *ICML*, 1999.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *ICML*, 2017.

Alexandre Péré, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration. In *ICLR*, 2018.

Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic Curiosity through Reachability. In *ICLR*, 2019.

Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *ICML*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075.

Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. Learning Goal Embeddings via Self-Play for Hierarchical Reinforcement Learning. *arXiv*, 2018a.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *ICLR*, 2018b.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, and Adam White. Horde : A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction Categories and Subject Descriptors. In *AAMAS*, 2011.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *NIPS*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *IEEE International Conference on Intelligent Robots and Systems*, 2012. ISSN 21530858.

David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised Control Through Non-Parametric Discriminative Rewards. In *ICLR*, 2019.

Rui Zhao and Volker Tresp. Curiosity-Driven Experience Prioritization via Density Estimation. In *NeurIPS Deep RL Workshop*, 2018.

A Formal Motivation

Here, we present a hypothesis relating *sibling rollouts* (that is, independently sampled rollouts using the same policy π, starting state s_0, and goal g) to the learning dynamics created by distance-to-goal shaped rewards $\tilde{r}$. We use the notation $\boldsymbol{\tau}^f, \boldsymbol{\tau}^c \sim \pi|g$ to denote a pair of trajectories from 2 sibling rollouts, where the superscript specifies that $\boldsymbol{\tau}^c$ ended closer to the goal than $\boldsymbol{\tau}^f$, i.e. that $\tilde{r}_{\boldsymbol{\tau}^c} \geq \tilde{r}_{\boldsymbol{\tau}^f}$. We use $\phi_g^\pi(\boldsymbol{\tau})$ to denote the probability that trajectory $\boldsymbol{\tau}$ earns a higher reward (i.e., is closer to the goal) than a sibling trajectory $\boldsymbol{\tau}'$

$$\phi_g^\pi(\boldsymbol{\tau}) = \mathbb{E}_{\boldsymbol{\tau}' \sim \pi|g} \left[1(\tilde{r}_{\boldsymbol{\tau}} > \tilde{r}_{\boldsymbol{\tau}'}) \right]. \tag{5}$$

This allows us to define the marginal (un-normalized) distributions for the sibling trajectories $\boldsymbol{\tau}^c$ and $\boldsymbol{\tau}^f$ as

$$\rho^c(\boldsymbol{\tau}|g) = \pi(\boldsymbol{\tau}|g) \cdot \phi_g^\pi(\boldsymbol{\tau}), \tag{6}$$

$$\rho^f(\boldsymbol{\tau}|g) = \pi(\boldsymbol{\tau}|g) \cdot (1 - \phi_g^\pi(\boldsymbol{\tau})), \tag{7}$$

where $\pi(\boldsymbol{\tau}|g) = \sum_{a,s \in \boldsymbol{\tau}} \pi(a|s,g)$.

Let us define the *pseudoreward* $\psi_g^\pi(\boldsymbol{\tau})$ as a simple scaling and translation of $\phi_g^\pi(\boldsymbol{\tau})$:

$$\psi_g^\pi(\boldsymbol{\tau}) = 2\phi_g^\pi(\boldsymbol{\tau}) - 1. \tag{8}$$

Importantly, since $\psi_g^\pi(\boldsymbol{\tau})$ captures how $\boldsymbol{\tau}$ compares to the distribution of trajectories induced by π, the policy can always improve its expected reward by increasing the probability of $\boldsymbol{\tau}$ for all $\boldsymbol{\tau}$ where $\psi_g^\pi(\boldsymbol{\tau}) > 0$ and decreasing the probability of $\boldsymbol{\tau}$ for all $\boldsymbol{\tau}$ where $\psi_g^\pi(\boldsymbol{\tau}) < 0$. Noting also that $\psi_g^\pi(\boldsymbol{\tau})$ increases monotonically with $\tilde{r}_{\boldsymbol{\tau}}$, we may gain some insight into the learning dynamics when optimizing $\tilde{r}_{\boldsymbol{\tau}}$ by considering the definition of the policy gradient for optimizing $\psi_g^\pi(\boldsymbol{\tau})$:

$$\nabla_\theta \mathbb{E}_{\boldsymbol{\tau} \sim \pi|g}[\psi_g^\pi(\boldsymbol{\tau})] = \mathbb{E}_{\boldsymbol{\tau} \sim \pi|g}[\nabla_\theta \log \pi(\boldsymbol{\tau}|g) \cdot (2\phi_g^\pi(\boldsymbol{\tau}) - 1)] \tag{9}$$

$$= \int \nabla_\theta \log \pi(\boldsymbol{\tau}|g) \cdot \left[\rho^c(\boldsymbol{\tau}|g) - \rho^f(\boldsymbol{\tau}|g) \right] d\boldsymbol{\tau}, \tag{10}$$

where θ is the set of internal parameters governing π.

This comparison exposes the importance of the quantity $\rho^c(\boldsymbol{\tau}|g) - \rho^f(\boldsymbol{\tau}|g)$, suggesting that the gradients should serve to reinforce trajectories where this difference is maximized. In other words, this suggests that optimizing π with respect to $\tilde{r}$ will concentrate the policy around trajectories that are *over* represented in $\rho^c(\boldsymbol{\tau}|g)$ and *under* represented in $\rho^f(\boldsymbol{\tau}|g)$.

While we cannot measure the marginal probabilities $\rho^c(\boldsymbol{\tau}|g)$ and $\rho^f(\boldsymbol{\tau}|g)$ for a given trajectory $\boldsymbol{\tau}$, we can sample trajectories from these distributions via sibling rollouts. Sibling Rivalry applies the interpretation that samples from $\rho^c(\boldsymbol{\tau}|g)$ (i.e., closer-to-goal siblings) capture the types of trajectories that the policy will converge towards when optimizing the distance-to-goal reward. By both limiting the use of $\boldsymbol{\tau}^c$ trajectories when computing gradients and encouraging trajectories to avoid the terminal states achieved by their sibling, we can counteract the learning dynamics that cause the policy to converge to a local optimum.

B Additional Examples

To further illustrate the behavior of Sibling Rivalry we profile learning in two simplified versions of the 2D point maze environment: an elongated corridor and a U-shaped hallway (Figure 6). In each, the agent starts at one end and must reach a goal location at the other end of the "maze." These two variants allow us to examine any tension between the distance-to-goal and distance-from-antigoal (i.e. distance from sibling terminal state) components of the reward r' used by SR.

In other words: **what is the trade-off between avoiding local optima created by the distance function and pursuing the global optimum created by the distance function?**

We address this question by comparing performance in corridor mazes and U-shaped mazes of varying side lengths (Figure 6, Top & Middle). In the corridor maze, the distance-to-goal signal creates a single global optimum that the agent should pursue. In the U-shaped maze, the distance-to-goal

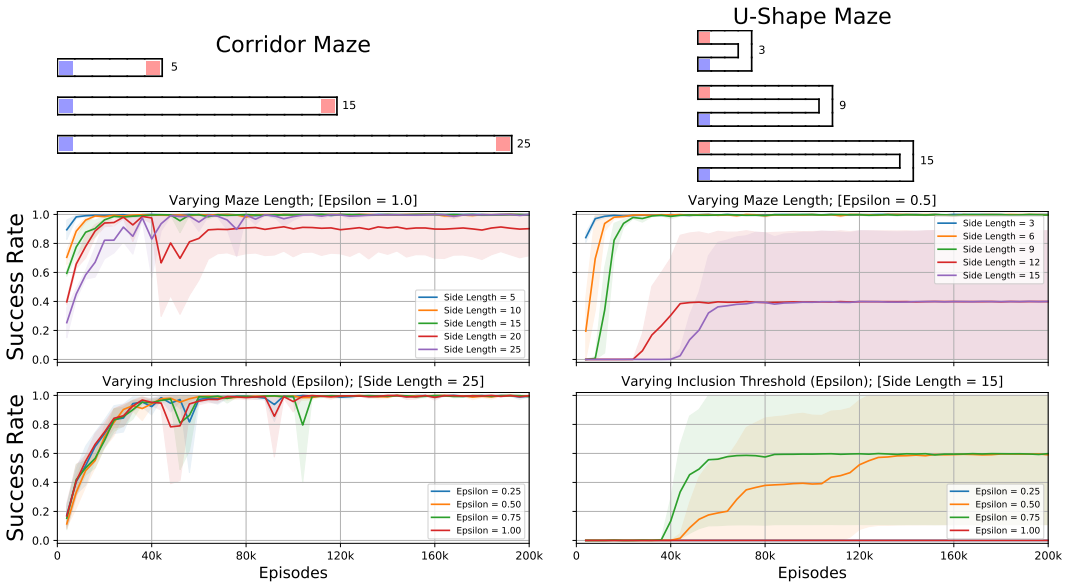

Figure 6: **Performance and limitations of Sibling Rivalry in toy 2D point mazes.** (Top) Corridor mazes (left) and U-shaped mazes (right) of varying side lengths. The number to the right of the maze indicates the side length. The start/goal location is sampled within the blue/red squares, respectively. (Middle) Performance of SR for each of the tested side lengths. (Bottom) Performance at varying settings of ϵ, the inclusion hyperparameter, on the longest variant of the corridor (left) and U-shaped (right) mazes. Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean$\pm$SD, clipped to [0, 1]).

signal creates a local optimum (in addition to the global optimum) that the agent must avoid. At the longest side-length tested, nearly all points within the U-shaped maze yield a worse distance-to-goal reward than the point at the local optimum, making the local optimum very difficult to escape. It is worth noting here that curiosity (ICM) and HER were observed to fail on both of these maze variants for the longest tested side lengths.

As shown in Figure 6, SR quickly solves the corridor maze, where the distance-to-goal signal serves as a good shaped reward. This result holds at the longest corridor setting for each of the ϵ settings tested (Figure 6, Bottom). Note: these settings correspond to fairly aggressive exclusion of the closer-to-goal sibling rollout. This simple environment offers a clear demonstration that SR preserves the distance-to-goal reward signal. However, as discussed in Section C, using an overly aggressive ϵ can lead to worse performance in a more complex environment.

Importantly, SR also solves the U-shaped variants, which are characterized by severe local optima. However, while we still observe decent performance for the most difficult versions of the U-shaped maze, this success depends strongly on a carefully chosen setting for ϵ. *As the distance function becomes increasingly misaligned with the structure of the environment, the range of good values for ϵ shrinks.* Section C provides further empirical insight into the influence of ϵ.

The combined observations from the corridor and U-shaped mazes illustrate that Sibling Rivalry achieves a targeted disruption of the learning dynamics associated with (non-global) local optima. This explains why SR does not interfere with solving the corridor maze, where local optima are not an issue, while being able to solve the U-shaped maze, characterized by severe local optima. Furthermore, these observations underscore that using r' and sibling rollouts for reward re-labeling automatically tailors the reward signal to the environment/task being solved.

C Controlling the Inclusion Hyperparameter

Sibling Rivalry makes use of a single hyperparameter ϵ to set the distance threshold for when to include the closer-to-goal trajectory $\boldsymbol{\tau}^c$ in the parameter updates. When $\epsilon = 0$, $\boldsymbol{\tau}^c$ is only included if

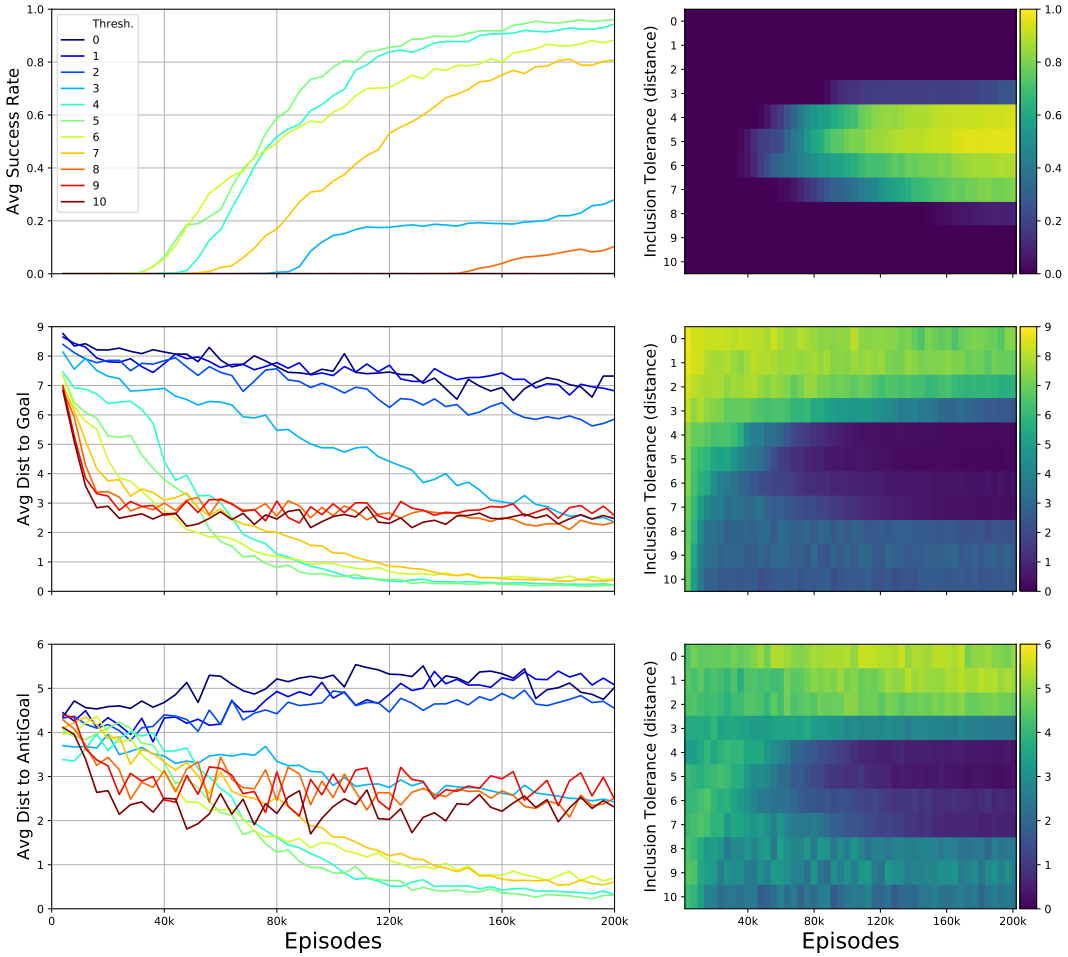

Figure 7: **Effect of Inclusion Threshold ϵ on Sibling Rivalry.** We re-run the 2D point maze experiments using SR with each of the ϵ settings shown. Rows report success rate, distance to goal, and distance to anti-goal (that is, distance between sibling rollouts) across training for each of the settings. Line plots and heatmap plots provide different views of the same data. This analysis identifies roughly 3 modes of behavior exhibited by our method in this environment. The first, over-exploration, occurs for the lower range of ϵ, where closer-to-goal trajectories are more aggressively discarded. Close inspection shows slow progress towards the goal and a tendency to increase inter-sibling distance (the latter trend appears to reverse near the end of the training window). The second mode corresponds to successful behavior: the agent can exploit the distance-to-goal signal but maintains enough diversity in its state distribution to avoid commitment to local optima. The third mode, under-exploration, occurs for the higher range of ϵ, where inclusion of the closer-to-goal trajectory is more permissive. These settings lead the agent to the same pitfall that prevents learning from naive distance-to-goal shaped rewards. That is, it quickly identifies a low-distance local optimum (consistently, the top corridor of the maze) and does not sufficiently explore in order to find a higher-reward region of the maze.

it reaches the goal. Conversely, when $\epsilon = \texttt{Inf}$, the algorithm always uses both trajectories (while still encouraging diversity through the augmented reward r'). We find that this parameter can be used to tune learning towards exploration or exploitation (of the distance-to-goal reward).

This is most evident in the impact of ϵ on learning progress in the 2D point maze environment, where local optima are numerous (and, in our observation, learning progress is most sensitive to ϵ). For the sake of demonstration, we performed a set of experiments for each of $\epsilon \in [0, 1, ...10]$ distance units. The 2D point maze itself is 10x10, giving us good coverage of options one might consider for ϵ in this environment. Interestingly, we observe three modes of the algorithm: over-exploration

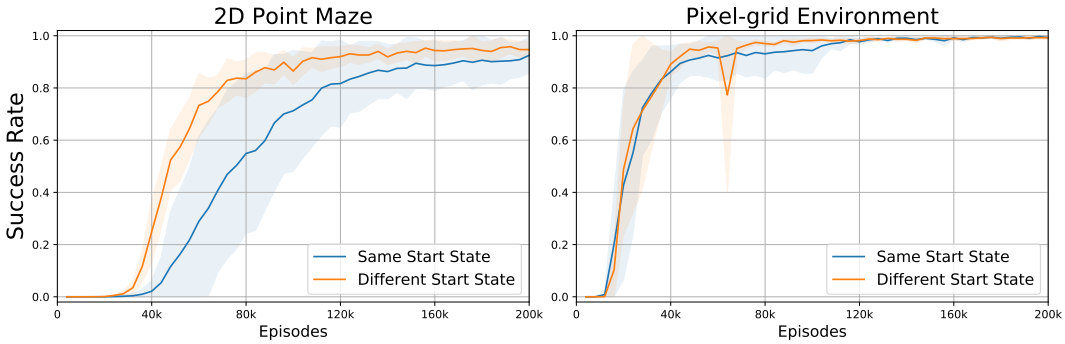

Figure 8: **Robustness to Different Starting States.** Performance of Sibling Rivalry when sibling rollouts share the same starting state s_0 versus independently sampled starting states. Curves show training progress for the 2D point maze task (left) and for the pixel-grid bit flipping task (right), averaged over 5 experiments (shaded area shows mean$\pm$SD, clipped to [0, 1]). Blue curves (same start state) follow the definition of 'sibling rollout' used in the main text.

(ϵ too low), successful learning, and under-exploration (ϵ too high). We observe these modes to be clearly identifiable using the metrics reported in Figure 7. In practice a much coarser search over this hyperparameter should be sufficient to identify the optimal range.

D Robustness to Different Starting States

Since some learning settings do not offer direct control over the starting state of an episode, we test the performance of Sibling Rivalry when the start states of sibling rollouts are sampled independently (Figure 8). For the 2D point maze environment, start locations are sampled independently from within the bottom left corner of the maze. For the pixel-grid environment, sibling rollouts use independently sampled grid locations as the starting position. In both cases, the siblings' starting states correspond to 2 independent samples from the task's underlying start state distribution. We compare performance under these sampling conditions to performance when the sibling rollouts use the same starting state. Interestingly, we observe faster convergence with independent starting states for the 2D point maze and roughly similar performance for the pixel-grid environment. These results suggest that Sibling Rivalry is robust to noise in the starting state and may even benefit from it. However, this is not an exhaustive analysis and one might expect different outcomes for environments where the policy tends to find different local optima depending on the episode's starting state. Nevertheless, these results indicate that Sibling Rivalry can be applied in settings where exact control over s_0 is not feasible.

E Comparison to Grid Oracle Baseline

We compare the performance of Sibling Rivalry to a *Grid Oracle* baseline (Savinov et al., 2019). The Grid Oracle augments the end-of-episode reward with a value proportional to the number of regions visited during the episode, computed by dividing the XY-space of the environment into a grid of discrete regions (with the number of divisions serving as the main hyperparameter). The Grid Oracle only sees the sparse reward plus the region-visitation reward. This baseline encourages the agent to cover a broad area, which, based on the exploration challenge presented by the maze environments, may act as a generically useful shaped reward for helping to discover the sparse reward from reaching the goal. Using the Grid Oracle shaped reward indeed facilitates discovery of the goal but can suffer from the fact that maximizing coverage within an episode does not guarantee task-useful behavior (Figure 9). Interestingly, we find that SR tends to more consistently solve the 2D point maze and ant maze tasks. It is also worth noting that SR only slightly lags the Grid Oracle baseline in terms of sample complexity. SR encourages both efficient and task-useful exploration by taking advantage of the properties of sibling rollouts and distance-based rewards.

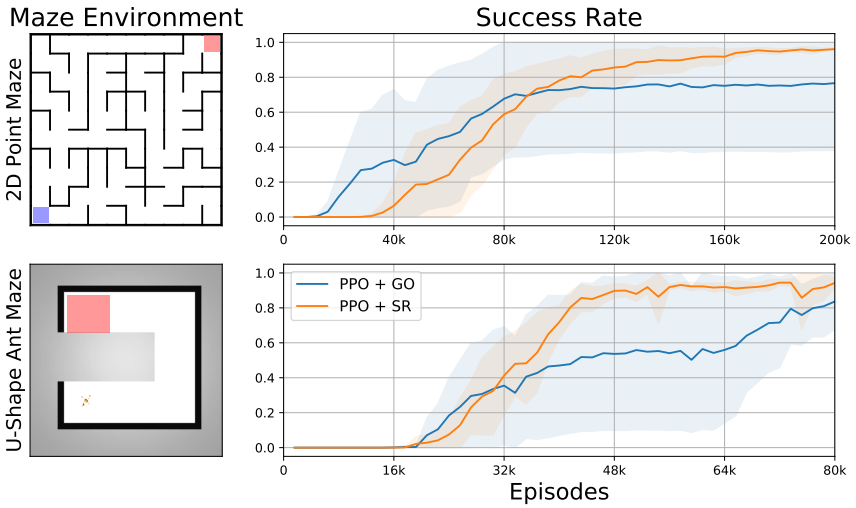

Figure 9: **Comparison with Grid Oracle baseline.** We compare performance of SR in the maze environments to performance when using a Grid Oracle reward. SR (PPO + SR) employs the self-balancing shaped reward r' (described in the main text), whereas the Grid Oracle (PPO + GO) adds a shaped reward based on the number of discrete environment regions visited within an episode. For the 2D maze, the start location is sampled within the blue square; in the ant maze, the agent starts near its pictured location. For both, the goal is randomly sampled from within the red square region. Lines show rate of goal completion averaged over 5 experiments (shaded area shows mean$\pm$SD, clipped to [0, 1]).

F Implementation Details and Experimental Hyperparameters

Here, we provide a more detailed description of the environments, tasks, and training implementations used in our experiments (Section 4). We first provide a general description of the training algorithms as they pertain to our experiments. We follow with task-specific details for each of the environments.

For all experiments, we distribute rollout collection over 20 parallel threads. Quantities regarding rollouts, epochs, and minibatches are all reported *per worker*.

Proximal Policy Optimization (PPO). Many of the experiments we perform use PPO as the backbone learning algorithm. We focus on PPO because of its strong performance and because it is well suited for the constraints imposed by the application of Sibling Rivalry. Specifically, our method requires the collection of multiple full rollouts in between network updates. PPO handles this well as it is able to make multiple updates from a large batch of transitions. While experimental variants that do not use SR do not require scheduling updates according to full rollouts, we do so for ease of comparison. The general approach we employ cycles between collection of full trajectories and multiple optimization epochs over minibatches of transitions within those trajectories. We apply a constant number of optimization epochs and updates per epoch, varying the sizes of the minibatches as needed based on the variable length of trajectories (due to either episode termination after goal-reaching or trajectory exclusion when using SR). We confirmed that this modification of the original algorithm did not meaningfully affect learning.

We standardize our PPO approach as much as possible to avoid results due to edge-case hyperparameter configurations, using manual search to identify such generally useful parameter settings. In the ant maze task, this standardized approach applies specifically to training the high-level policy. We also use PPO to train the low-level policy but adopt a more specific approach for that based on its unique role in our experiments (described below).

For PPO variants, the output head of the policy network specifies the $\alpha \in \mathbb{R}^2$ and $\beta \in \mathbb{R}^2$ control parameters of a Beta distribution to allow sampling actions within a truncated range (Chou et al., 2017). We shift and scale the sampled values to correspond to the task action range. We also include entropy regularization to prevent the policy from becoming overly deterministic early during training.

Table 1: Implementation details for experiments using PPO

	Point maze			Ant maze (high)			Bit flipping		
Hyperparameter	PPO	+SR	+ICM	PPO	+SR	+ICM	PPO	+SR	+ICM
Rollouts per Update	4			4			4		
Epochs per Update	4			2			4		
m.Batches per Epoch	4			4			4		
Learning Rate (LR)	0.001			0.001			0.001		
LR Decay	0.999			1.0			0.999		
Entropy Reg λ	0.025			0.025			0.025	0.0	0.025
GAE λ	0.98			0.98			0.98		
Bootstrap Value	N		Y	N		Y	N		Y
Discount Factor	1.0		0.98	1.0		0.98	1.0		0.98
Inclusion thresh. (ϵ)		5.0			10.0			Inf	

Table 2: Implementation details for off-policy experiments

Hyperparameter	Point maze	Ant maze (high)	Bit flipping
Rollouts per Update	4		
m.Batches per Update	40		
m.Batches size	64	128	128
Learning Rate (LR)	0.001		
Action L_2 λ	0.25	0.0002	NA
Behavior action noise	$0.1 \times$ action range		NA
Behavior action epsilon	0.2		
Polyak coefficient	0.95		
Bootstrap Value	Y		
Discount Factor	0.98		

Intrinsic Curiosity Module (ICM). We base our implementation of ICM off the guidelines provided in Burda et al. (2018a). We weigh the curiosity-driven intrinsic reward by 0.01 compared to the sparse reward. Note that in the settings we used, ICM is only accompanied by sparse extrinsic rewards, meaning that it only experiences the intrinsic rewards until it (possibly) discovers the goal region. During optimization, we train the curiosity network modules (whose architectures follow similar designs to the policy and value for the given task) at a rate of 0.05 compared to the policy and value network modules.

2D point maze navigation. The 2D point maze is implemented in a 10x10 environment (arbitrary units) consisting of an array of pseudo-randomly connected 1x1 squares. The construction of the maze ensures that all squares are connected to one another by exactly one path. This is a continuous environment. The agent sees as input its 2D coordinates and well as the 2D goal coordinates, which are always somewhere near the top right corner of the maze. The agent takes an action in a 2D space that controls the direction and magnitude of the step it takes, with the outcome of that step potentially affected by collisions with walls. The agent does not observe the walls directly, creating a difficult exploration environment. For all experiments, we learn actor and critic networks with 3 hidden layers of size 128 and `ReLU` activation functions.

Table 3: Environment details

Setting	$S \in$	$G \in$	$A \in$
Point maze	$\mathbb{R}^2$	$\mathbb{R}^2$	$[-0.95, 0.95]^2$
Ant maze (high)	$\mathbb{R}^{30}$	$\mathbb{R}^2$	$[-5, 5]^2$
Ant maze (low)	$\mathbb{R}^{30}$	$\mathbb{R}^2$	$[-30, 30]^8$
Bit flipping	$\{0, 1\}^{13\times13\times2}$	$\{0, 1\}^{13\times13}$	$\{0...9\}$
Minecraft	$s^v \in \mathbb{R}^{80\times120\times3}$, $s^c \in \{0...N_b\}^{11\times11\times3}$	$\{0...N_b\}^{11\times11\times3}$	$\{0...20\}$

Table 4: Task details

Setting	$m(s)$	$d(,)$	δ	Max. T
Point maze	$\mathbb{I}$	L_2	0.15	50
Ant maze (high)	$[s^0, s^1]$	L_2	1.0	25 (=500 env steps)
Ant maze (low)	$[s^0, s^1]$	L_2	NA	20 (env steps)
Bit flipping	$s^{:,:,0}$	L_1	0.0	50
Minecraft	s^c	$\sum x_{ijk} \neq y_{ijk}$	0.0	100

Ant maze navigation with hierarchical RL. The ant maze experiment borrows a similar set up to the point maze but trades complexity of the maze for complexity in the navigation behavior. We use this as a lens to study how the different algorithms handle HRL in this setting. We divide the agent into a high-level and low-level policy, wherein the high-level policy proposes subgoals and the low-level agent is rewarded for reaching those subgoals. For all experiments, we allow the high-level policy to propose a new subgoal g^L every 20 environment timesteps. From the perspective of training the low-level policy, we treat each such 20 steps with a particular subgoal as its own mini-episode. At the end of the full episode, we perform 2 epochs of PPO training to improve the low-level policy, using distance-to-subgoal as the reward.

The limits of the maze are $[-4, 20]$ in both height and width. The agent starts at position $(0, 0)$ and must navigate to goal location $g = (x_g, y_g)$ with coordinates sampled within the range of $x_g \in [-3.5, 3.5]$ and $y_g \in [12.5, 19.5]$. It should be noted that, compared to previous implementations of this environment and task (Nachum et al., 2018), we do not include the full range of the maze in the distribution of task goals. For the agent to ever see the sparse reward, it must navigate from one end of the U-maze to the other and cannot bootstrap this exploration by learning from goals that occur along the way. As one might expect, the learning problem becomes considerably easier when this broad goal distribution is used; we experiment in the more difficult setting since we do not wish to impose the assumption that a task's goal distribution will naturally tile goals from ones that are trivially easy to reach to those that are difficult.

At timestep t, the high-level controller outputs a 2-dimensional action $a_t \in [-5, 5]^2$, which is used to compute the subgoal $g^L_t = m(s_t) + a_t$. In other words, the high-level action specifies the relative coordinates the low-level policy should achieve. From the perspective of training the high-level policy, we only consider the timesteps where it takes an action and consider the result produced by the low-level policy as the effect of having taken the high-level action.

In all experiments, both the high- and low-level actor and critic networks use 3 hidden layers of size 128 and `ReLU` activation functions.

2D bit flipping task. We extend the bit flipping example used to motivate HER (Andrychowicz et al., 2017) to a 2D environment in which interaction with the bit array depends on location. In this setting, the agent begins at a random position on a 13x13 grid with none of its bit array switched on. Its goal is to reproduce the bit array specified by the goal. To populate these examples, we

procedurally generate goal arrays by simulating a simple agent that changes direction every few steps and toggles bits it encounters along the way.

We include this example mostly to illustrate (i) that our method can work in this entirely discrete learning setting and (ii) that naive distance-to-goal based rewards are exceptionally prone to even brittle local optima, such as the ones created when the agent learns to avoid taking the toggle-bit action.

We report the (eventually) successful performance using vanilla DQN but point out that this required modifying the reward delivery for this particular agent. In all previous settings, agents trained on shaped rewards receive that reward only at the end of the episode (and no discounting is used). While it is beyond the scope of this work to decipher this observation, we found that DQN could only learn if the shaped reward was exposed at every time step (using a discounting of $\gamma = 0.98$). The variant that used the reward-at-end scheme never learned.

For all bit flipping experiments, we use 2D convolution to encode the states and goals. We pool the convolution output with `MaxPooling`, apply `LayerNorm`, and finally pass the hidden state through a fully connected layer to get the actor and critic outputs.

3D construction in Minecraft. To test our proposed method at a more demanding scale, we implement a custom structure-building task in Minecraft using the Malmo platform. In this task, we place the agent at the center of a "build arena" which is populated in one of several full Minecraft worlds. In this particular setting, the agent has no task-specific incentive to explore the outer world but is free to do so. Our task requires the agent to navigate the arena and control its view and orientation in order to reproduce the structure provided as a goal (similar to a 3D version of the bit flipping example but with richer mechanics and more than one type of block that can be placed). All goals specify a square structure made of a single block type that is either 1 or 2 blocks high with corners at randomly chosen locations in the arena. For each sampled goal, we randomly choose those configuration details and keep the sampled goal provided that it has no more than 34 total blocks (to ensure that the structure can be completed within a 100 timestep episode). The agent begins each episode with the necessary inventory to accomplish the goal. Specifically, the goal structures are always composed of 1 of 3 block types and the agent always begins with 64 blocks of each of those types. It may place other block types if it finds them.

The agent is able to observe the first-person visual input of the character it controls as well as the 3D cuboid of the goal structure and the 3D cuboid of the current build arena. The agent therefore has access to the structure it has accomplished but must also use the visual input to determine the next actions to direct further progress.

The visual input is process through a shallow convolution network. Similarly, the cuboids, which are represented as 3D tensors of block-type indices, are embedded through a learned lookup and processed via 3D convolution. The combined hidden states are used as inputs to the policy network. The value network uses separate weights for 3D convolution (since it also takes the anti-goal cuboid as input) but shares the visual encoder with the policy.

Owing to the computational intensity and long run-time of these experiments, we limit our scope to the demonstration of Sibling Rivalry in this setting. However, we do confirm that, like with the bit flipping example, naive distance-to-goal reward shaping fails almost immediately (the agent learns to never place blocks in the arena within roughly 1000 episodes).

For the work presented here, we compute the reward as the change in the distance produced by placing a single block (and use discounting of $\gamma = 0.99$). We find that this additional densification of the reward signal produces faster training in this complex environment.