# Prioritized Level Replay

**Minqi Jiang** [1 2] **Edward Grefenstette** [1 2] **Tim Rocktäschel** [1 2]

## Abstract

Environments with procedurally generated content serve as important benchmarks for testing systematic generalization in deep reinforcement learning. In this setting, each *level* is an algorithmically created environment instance with a unique configuration of its factors of variation. Training on a prespecified subset of levels allows for testing generalization to unseen levels. What can be learned from a level depends on the current policy, yet prior work defaults to uniform sampling of training levels independently of the policy. We introduce *Prioritized Level Replay* (PLR), a general framework for selectively sampling the next training level by prioritizing those with higher estimated learning potential when revisited in the future. We show TD-errors effectively estimate a level's future learning potential and, when used to guide the sampling procedure, induce an emergent curriculum of increasingly difficult levels. By adapting the sampling of training levels, PLR significantly improves sample-efficiency and generalization on Procgen Benchmark—matching the previous state-of-the-art in test return—and readily combines with other methods. Combined with the previous leading method, PLR raises the state-of-the-art to over 76% improvement in test return relative to standard RL baselines.

## 1. Introduction

Deep reinforcement learning (RL) easily overfits to training experiences, making generalization a key open challenge to widespread deployment of these methods. Procedural content generation (PCG) environments have emerged as a promising class of problems with which to probe and address this core weakness (Risi & Togelius, 2020; Chevalier-Boisvert et al., 2018; Cobbe et al., 2020a; Juliani et al., 2019; Zhong et al., 2020; Küttler et al., 2020). Unlike singleton environments, like the Arcade Learning Environment games (Bellemare et al., 2013), PCG environments take on algorithmically created configurations at the start of each training episode, potentially varying the layout, asset appearances, or even game dynamics. Each environment instance generated this way is called a *level*. In mapping an identifier, such as a random seed, to each level, PCG environments allow us to measure a policy's generalization to held-out test levels. In this paper, we assume only a blackbox generation process that returns a level given an identifier. We avoid the strong assumption of control over the generation procedure itself, explored by several prior works (see Section 6). Further, we assume levels follow common latent dynamics, so that in aggregate, experiences collected in individual levels reveal general rules governing the entire set of levels.

Despite its humble origin in games, the PCG abstraction proves far-reaching: Many control problems, such as teaching a robotic arm to stack blocks in a specific formation, easily conform to PCG. Here, each level may consist of a unique combination of initial block positions, arm state, and target formation. In a vastly different domain, Hanabi (Bard et al., 2020) also conforms to PCG, where levels map to initial deck orderings. These examples illustrate the generality of the PCG abstraction: Most challenging RL problems entail generalizing across instances (or levels) differing along some underlying factors of variation and thereby can be aptly framed as PCG. This highlights the importance of effective deep RL methods for PCG environments.

Many techniques have been proposed to improve generalization in the PCG setting (see Section 6), requiring changes to the model architecture, learning algorithm, observation space, or environment structure. Notably, these approaches default to uniform sampling of training levels. We instead hypothesize that the variation across levels implies that at each point of training, each level likely holds different potential for an agent to learn about the structure shared across levels to improve generalization. Inspired by this insight and selective sampling methods from active learning, we investigate whether sampling the next training level weighed by its learning potential can improve generalization.

We introduce Prioritized Level Replay (PLR), illustrated in Figure 1, a method for sampling training levels that exploits

[1]Facebook AI Research, London, United Kingdom [2]University College London, London, United Kingdom. Correspondence to: Minqi Jiang <msj@fb.com>.

*Proceedings of the $38^{th}$ International Conference on Machine Learning*, PMLR 139, 2021. 

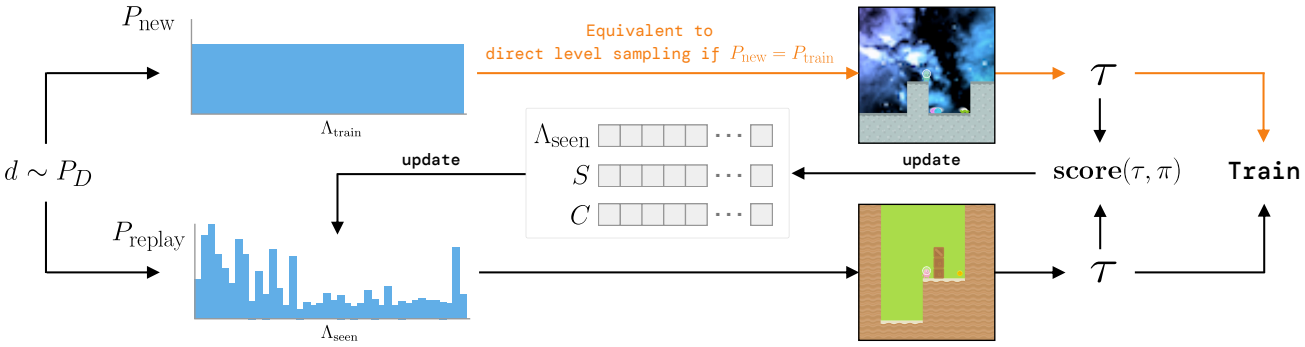


*Figure 1.* Overview of Prioritized Level Replay. The next level is either sampled from a distribution with support over unseen levels (top), which could be the environment's (perhaps implicit) full training-level distribution, or alternatively, sampled from the replay distribution, which prioritizes levels based on future learning potential (bottom). In either case, a trajectory $\tau$ is sampled from the next level and used to update the replay distribution. This update depends on the lists of previously seen levels $\Lambda_{\text{seen}}$, their latest estimated learning potentials $S$, and last sampled timestamps $C$.

the differences in learning potential among levels to improve both sample efficiency and generalization. PLR selectively samples the next training level based on an estimated learning potential of replaying each level anew. During training, our method updates scores estimating each level's learning potential as a function of the agent's policy and temporal-difference (TD) errors collected along the last trajectory sampled on that level. Our method then samples the next training level from a distribution derived from a normalization procedure over these level scores. PLR makes no assumptions about how the policy is updated, so it can be used in tandem with any RL algorithm and combined with many other methods such as data augmentation. Our method also does not assume any external, predefined ordering of levels by difficulty or other criteria, but instead derives level scores dynamically during training based on how the policy interacts with the environment. The only requirements are as follows—satisfied by almost any problem that can be framed as PCG, including RL environments implemented as seeded simulators: (i) Some notion of "level" exists, such that levels follow common latent dynamics; (ii) such levels can be sampled from the environment in an identifiable way; and (iii) given a level identifier, the environment can be set to that level to collect new experiences from it.

While previous works in off-policy RL devised effective methods to directly reuse *past* experiences for training (Schaul et al., 2016; Andrychowicz et al., 2017), PLR uses past experiences to inform the collection of *future* experiences by estimating how much replaying each level anew will benefit learning. Our method can thus be seen as a forward-view variation of prioritized experience replay, and an online counterpart to this off-policy method.

This paper makes the following contributions[1]: (i) We introduce a computationally-efficient algorithm for sampling levels during training based on an estimate of the future learning potential of collecting new experiences from each level; (ii) we show our method significantly improves generalization on 10 of 16 environments in Procgen Benchmark and two challenging MiniGrid environments; (iii) we demonstrate our method combines with the previous leading method to set a new state-of-the-art on Procgen Benchmark; and (iv) we show our method induces an implicit curriculum over training levels in sparse reward settings.

## 2. Background

In this paper, we refer to a *PCG environment* as any computational process that, given a level identifier (e.g. an index or a random seed), generates a *level*, defined as an environment instance exhibiting a unique configuration of its underlying factors of variation, such as layout, asset appearances, or specific environment dynamics (Risi & Togelius, 2020). For example, MiniGrid's MultiRoom environment instantiates mazes with varying numbers of rooms based on the seed (Chevalier-Boisvert et al., 2018). We refer to sampling a new trajectory generated from the agent's latest policy acting on a given level $l$ as *replaying* that level $l$.

The level diversity of PCG environments makes them useful testbeds for studying the robustness and generalization ability of RL agents, measured by agent performance on unseen test levels. The standard test evaluation protocol for PCG environments consists of training the agent on a finite number of training levels $\Lambda_{\text{train}}$, and evaluating performance on unseen test levels $\Lambda_{\text{test}}$, drawn from the set of all levels. Training levels are sampled from an arbitrary distribution $P_{\text{train}}(l|\Lambda_{\text{train}})$. We call this training process *direct level sampling*. A common variation of this protocol sets $\Lambda_{\text{train}}$ to the set of all levels, though in practice, the agent will still only effectively see a finite set of levels after training for a finite

[1]Our code is available at https://github.com/facebookresearch/level-replay.

number of steps. In the case of a finite training set, typically $P_{\text{train}}(l|\Lambda_{\text{train}}) = \textbf{Uniform}(l; \Lambda_{\text{train}})$. See Appendix C for the pseudocode outlining this procedure.

PCG environments naturally lend themselves to curriculum learning. Prior works have shown that directly altering levels to match their difficulty to the agent's abilities can improve generalization (Justesen et al., 2018; Dennis et al., 2020; Chevalier-Boisvert et al., 2018; Zhong et al., 2020). These findings further suggest the levels most useful for improving an agent's policy vary throughout the course of training. In this work, we consider how to automatically discover a curriculum that improves generalization for a general black-box PCG environment—crucially, without assuming any knowledge or control of how levels are generated (beyond providing the random seed or other indicial level identifier).

## 3. Prioritized Level Replay

In this section, we present *Prioritized Level Replay* (PLR), an algorithm for selectively sampling the next training level given the current policy, by prioritizing levels with higher estimated learning potential when replayed (that is, revisited). PLR is a drop-in replacement for the experience-collection process used in a wide range of RL algorithms. Algorithm 1 shows how it is straightforward to incorporate PLR into a generic policy-gradient training loop. For clarity, we focus on training on batches of complete trajectories (see Appendix C for pseudocode of PLR with $T$-step rollouts).

Our method, illustrated in Figure 1 and fully specified in Algorithm 2, induces a dynamic, nonparametric sampling distribution $P_{\text{replay}}(l|\Lambda_{\text{seen}})$ over previously visited training levels $\Lambda_{\text{seen}}$ that prioritizes visited levels with higher learning potential based on properties of the agent's past trajectories. We refer to $P_{\text{replay}}(l|\Lambda_{\text{seen}})$ as the *replay distribution*. Throughout training, our method updates this replay distribution according to a heuristic score, assigning greater weight to visited levels with higher *future* learning potential. Using dynamic arrays $S$ and $C$ of equal length to $\Lambda_{\text{seen}}$, PLR tracks level scores $S_i \in S$ for each visited training level $l_i$ based on the latest episode trajectory on $l_i$, as well as the episode count $C_i \in C$ at which each level $l_i \in \Lambda_{\text{seen}}$ was last sampled. Our method updates $P_{\text{replay}}$ after each terminated episode by computing a mixture of two distributions, $P_S$, based on the level scores, and $P_C$, based on how long ago each level was last sampled:

$$P_{\text{replay}} = (1-\rho) \cdot P_S + \rho \cdot P_C, \tag{1}$$

where the staleness coefficient $\rho \in [0, 1]$ is a hyperparameter. We discuss how we compute level scores $S_i$, parameterizing the scoring distribution $P_S$, and the staleness distribution $P_C$, in Sections 3.1 and 3.2, respectively.

PLR chooses the next level at the start of every training episode by first sampling a replay-decision from a

**Algorithm 1** Policy-gradient training loop with PLR

**Input:** Training levels $\Lambda_{\text{train}}$, policy $\pi_\theta$,
policy update function $\mathcal{U}(\mathcal{B}, \theta) \to \theta'$, and batch size $N_b$.
Initialize level scores $S$ and level timestamps $C$
Initialize global episode counter $c \leftarrow 0$
Initialize the ordered set of visited levels $\Lambda_{\text{seen}} = \varnothing$
Initialize experience buffer $\mathcal{B} = \varnothing$
**while** training **do**
  $\mathcal{B} \leftarrow \varnothing$
  **while** collecting experiences **do**
    $\mathcal{B} \leftarrow \mathcal{B} \cup$ **collect_experiences**$(\Lambda_{\text{train}}, \Lambda_{\text{seen}}, \pi_\theta, S, C, c)$
    *Using Algorithm 2*
  **end while**
  $\theta \leftarrow \mathcal{U}(\mathcal{B}, \theta)$ *Update policy using collected experiences*
**end while**

**Algorithm 2** Experience collection with PLR

**Input:** Training levels $\Lambda_{\text{train}}$, visited levels $\Lambda_{\text{seen}}$, policy $\pi$,
global level scores $S$, global level timestamps $C$, and
global episode counter $c$.
**Output:** A sampled trajectory $\tau$
$c \leftarrow c + 1$
Sample replay decision $d \sim P_D(d)$
**if** $d = 0$ **and** $|\Lambda_{\text{train}} \setminus \Lambda_{\text{seen}}| > 0$ **then**
  Define new index $i \leftarrow |S| + 1$
  Sample $l_i \sim P_{\text{new}}(l|\Lambda_{\text{train}}, \Lambda_{\text{seen}})$ *Sample an unseen level*
  Add $l_i$ to $\Lambda_{\text{seen}}$
  Add initial value $S_i = 0$ to $S$ and $C_i = 0$ to $C$
**else**
  Sample $l_i \sim (1-\rho) \cdot P_S(l|\Lambda_{\text{seen}}, S) + \rho \cdot P_C(l|\Lambda_{\text{seen}}, C, c)$
  *Sample a level for replay*
**end if**
Sample $\tau \sim P_\pi(\tau|l_i)$
Update score $S_i \leftarrow \textbf{score}(\tau, \pi)$ and timestamp $C_i \leftarrow c$

Bernoulli (or similar) distribution $P_D(d)$ to determine whether to replay a level sampled from the replay distribution $P_{\text{replay}}(l|\Lambda_{\text{seen}})$ or to experience a new, unseen level from $\Lambda_{\text{train}}$, according to some distribution $P_{\text{new}}(l|\Lambda_{\text{train}} \setminus \Lambda_{\text{seen}})$. In practice, for the case of a finite number of training levels, we implement $P_{\text{new}}$ as a uniform distribution over the remaining unseen levels. For the case of a countably infinite number of training levels, we simulate $P_{\text{new}}$ by sampling levels from $P_{\text{train}}$ until encountering an unseen level. In our experiments based on a finite number of training levels, we opt to naturally anneal $P_D(d = 1)$ as $|\Lambda_{\text{seen}}|/|\Lambda_{\text{train}}|$, so replay occurs more often as more training levels are visited.

The following sections describes how Prioritized Level Replay updates the replay distribution $P_{\text{replay}}(l|\Lambda_{\text{seen}})$, namely through level scoring and staleness-aware prioritization.

### 3.1. Scoring Levels for Learning Potential

After collecting each complete episode trajectory $\tau$ on level $l_i$ using policy $\pi$, our method assigns $l_i$ a score $S_i = \textbf{score}(\tau, \pi)$ measuring the learning potential of replaying $l_i$ in the future. We employ a function of the TD-error at timestep $t$, $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, as a proxy

for this learning potential. The expectation of the TD-error over next states is equivalent to the advantage estimate, and therefore higher-magnitude TD-errors imply greater discrepancy between expected and actual returns, making $\delta_t$ a useful measure of the learning potential in revisiting a particular state transition. To prioritize the learning potential of future experiences resulting from replaying a level, we use a scoring function based on the *average magnitude* of the Generalized Advantage Estimate (GAE; Schulman et al., 2016) over each of $T$ time steps in the latest trajectory $\tau$ from that level:

$$S_i = \textbf{score}(\tau, \pi) = \frac{1}{T}\sum_{t=0}^{T}\left|\sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k\right|. \quad (2)$$

While the GAE at time $t$ is most commonly expressed as the discounted sum of all 1-step TD-errors starting at $t$ as in Equation 2, it is equivalent to an exponentially-discounted sum of all $k$-step TD-errors from $t$, with discount factor $\lambda$. By considering all $k$-step TD-errors, the GAE mitigates the bias introduced by the bootstrap term in 1-step TD-errors. The discount factor $\lambda$ then controls the trade-off between bias and variance. Our scoring function considers the absolute value of the GAE, as we assume the learning potential grows with the magnitude of the TD-error irrespective of its sign. This also avoids opposite signed errors canceling out.

Another useful interpretation of Equation 2 comes from observing that the GAE magnitude at $t$ is equivalent to the L1 value loss $|\hat{V}_t - V_t|$ under a policy-gradient algorithm that uses GAE for its own advantage estimates (and therefore value targets $\hat{V}_t$), as done in state-of-the-art implementations of PPO (Schulman et al., 2017) used in our experiments. Unless otherwise indicated, PLR refers to the instantiation of our algorithm with L1 value loss as the scoring function.

We further formulate the *Value Correction Hypothesis* to motivate our approach: In sparse reward settings, prioritizing the sampling of training levels with greatest average absolute value loss leads to a curriculum that improves both sample-efficiency and generalization. We reason that on threshold levels (i.e. those at the limit of the agent's current abilities) the agent will see non-stationary returns (or value targets)—and therefore incur relatively high value errors—until it learns to solve them consistently. In contrast, levels beyond the agent's current abilities tend to result in stationary value targets signaling failure and therefore low value errors, until the agent learns useful, transferable behaviors from threshold levels. Prioritizing levels by value loss then naturally guides the agent along the expanding threshold of its ability—without the need for any externally provided measure of difficulty. We believe that learning behaviors systematically aligned with the inherent complexities of the environment in this way may lead to better generalization, and will seek to verify this empirically in Section 5.2.

While we provide principled motivations for our specific choice of scoring function, we emphasize that in general, the scoring function can be any approximation of learning potential based on trajectory values. Note that candidate scoring functions should asymptotically decrease with frequency of level visitation to avoid mode collapse of $P_{\text{replay}}$ to a limited set of levels and possible overfitting. In Section 4, we compare our choice of the GAE magnitude, or equivalently, the L1 value loss, to alternative TD-error-based and uncertainty-based scoring approaches, listed in Table 1.

Given level scores, we use normalized outputs of a prioritization function $h$ evaluated over these scores and tuned using a temperature parameter $\beta$ to define the score-prioritized distribution $P_S(\Lambda_{\text{train}})$ over the training levels, under which

$$P_S(l_i|\Lambda_{\text{seen}}, S) = \frac{h(S_i)^{1/\beta}}{\sum_j h(S_j)^{1/\beta}}. \quad (3)$$

The function $h$ defines how differences in level scores translate into differences in prioritization. The temperature parameter $\beta$ allows us to tune how much $h(S)$ ultimately determines the resulting distribution. We make the design choice of using rank prioritization, for which $h(S_i) = 1/\text{rank}(S_i)$, where $\text{rank}(S_i)$ is the rank of level score $S_i$ among all scores sorted in descending order. We also experimented with proportional prioritization ($h(S_i) = S_i$) as well as greedy prioritization (the level with the highest score receives probability 1), both of which tend to perform worse.

### 3.2. Staleness-Aware Prioritization

As the scores used to parameterize $P_S$ are a function of the state of the policy at the time the associated level was last played, they come to reflect a gradually more off-policy measure the longer they remain without an update through replay. We mitigate this drift towards "off-policy-ness" by explicitly mixing the sampling distribution with a staleness-prioritized distribution $P_C$:

$$P_C(l_i|\Lambda_{\text{seen}}, C, c) = \frac{c - C_i}{\sum_{C_j \in C} c - C_j} \quad (4)$$

which assigns probability mass to each level $l_i$ in proportion to the level's *staleness* $c - C_i$. Here, $c$ is the count of total episodes sampled so far in training and $C_i$ (referred to as the level's timestamp) is the episode count at which $l_i$ was last sampled. By pushing support to levels with staler scores, $P_C$ ensures no score drifts too far off-policy.

Plugging Equations 3 and 4 into Equation 1 gives us a replay distribution that is calculated as

$$P_{\text{replay}}(l_i) = (1-\rho)\cdot P_S(l_i|\Lambda_{\text{seen}}, S) + \rho \cdot P_C(l_i|\Lambda_{\text{seen}}, C, c).$$

Thus, a level has a greater chance of being sampled when its score is high or it has not been sampled for a long time.

## 4. Experimental Setting

We evaluate PLR on several PCG environments with various combinations of scoring functions and prioritization schemes, and compare to the most common direct level sampling baseline of $P_{\text{train}}(l|\Lambda_{\text{train}}) = \mathbf{Uniform}(l; \Lambda_{\text{train}})$. We train and test on all 16 environments in the Procgen Benchmark on easy and hard difficulties, but focus discussion on the easy results, which allow direct comparison to several prior studies. We compare to UCB-DrAC (Raileanu et al., 2021), the state-of-the-art image augmentation method on this benchmark, and mixreg (Wang et al., 2020a), a recently introduced data augmentation method. We also compare to TSCL Window (Matiisen et al., 2020), which resembles PLR with an alternative scoring function using the slope of recent returns and no staleness sampling. For fair comparison, we also evaluate a custom TSCL Window variant that mixes in the staleness distribution $P_C$ weighted by $\rho > 0$. Further, to demonstrate the ease of combining PLR with other methods, we evaluate UCB-DrAC using PLR for sampling training levels. Finally, we test the Value Correction Hypothesis on two challenging MiniGrid environments.

We measure episodic *test returns* per game throughout training, as well as the performance of the final policy over 100 unseen test levels of each game relative to PPO with uniform sampling. We also evaluate the mean normalized episodic test return and mean generalization gap, averaged over all games (10 runs each). We normalize returns according to Cobbe et al. (2019) and compute the generalization gap as train returns minus test returns. Thus, a larger gap indicates more overfitting, making it an apt measure of generalization. We assess statistical significance at $p = 0.05$, using the Welch t-test.

In line with the standard baseline for these environments, all experiments use PPO with GAE for training. For Procgen, we use the same ResBlock architecture as Cobbe et al. (2020a) and train for 25M total steps on 200 levels on the easy setting as in the original baselines. For MiniGrid, we use a 3-layer CNN architecture based on Igl et al. (2019), and provide approximately 1000 levels of each difficulty per environment during training. Detailed descriptions of the environments, architectures, and hyperparameters used in our experiments (and how they were set or obtained) can be found in Appendix A. See Table 1 for the full set of scoring functions investigated in our experiments.

Additionally, we extend PLR to support training on an unbounded number of levels by tracking a rolling, finite buffer of the top levels so far encountered by learning potential. Appendix B.3 reports the results of this extended PLR algorithm when training on the full level distribution of the MiniGrid environments featured in our main experiments.

*Table 1.* Scoring functions investigated in this work.

| **Scoring function** | $\mathbf{score}(\tau, \pi)$ |
|---|---|
| Policy entropy | $\frac{1}{T}\sum_{t=0}^{T}\sum_{a}\pi(a,s_t)\log\pi(a,s_t)$ |
| Policy min-margin | $\frac{1}{T}\sum_{t=0}^{T}(\max_a \pi(a,s_t) - \max_{a\neq\max_a\pi(a,s_t)}\pi(a,s_t))$ |
| Policy least-confidence | $\frac{1}{T}\sum_{t=0}^{T}(1-\max_a\pi(a,s_t))$ |
| 1-step TD error | $\frac{1}{T}\sum_{t=0}^{T}\lvert\delta_t\rvert$ |
| GAE | $\frac{1}{T}\sum_{t=0}^{T}\sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k$ |
| GAE magnitude (L1 value loss) | $\frac{1}{T}\sum_{t=0}^{T}\left\lvert\sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k\right\rvert$ |

## 5. Results and Discussion

Our main findings are that (i) PLR significantly improves both sample efficiency and generalization, attaining the highest normalized mean test and train returns and mean reduction in generalization gap on Procgen out of all individual methods evaluated, while matching UCB-DrAC in test improvement relative to PPO; (ii) alternative scoring functions lead to inconsistent improvements across environments; (iii) PLR combined with UCB-DrAC sets a new state-of-the-art on Procgen; and (iv) PLR induces an implicit curriculum over training levels, which substantially aids training in two challenging MiniGrid environments.

### 5.1. Procgen Benchmark

Our results, summarized in Figure 2, show PLR with rank prioritization ($\beta = 0.1$, $\rho = 0.1$) leads to the largest statistically significant gains in mean normalized test and train returns and reduction in generalization gap compared to uniform sampling, outperforming all other methods besides UCB-DrAC + PLR. PLR combined with UCB-DrAC sees the most drastic improvements in these metrics. As reported in Table 2, UCB-DrAC + PLR yields a 76% improvement in mean test return relative to PPO with uniform sampling, and a 28% improvement relative to the previous state-of-the-art set by UCB-DrAC. While PLR with rank prioritization leads to statistically significant gains in test return on 10 of 16 environments and proportional prioritization, on 11 of 16 games, we prefer rank prioritization: While we find the two comparable in mean normalized returns, Figure 3 shows rank prioritization results in higher mean *unnormalized* test returns and a significantly lower mean generalization gap, averaged over all environments.

Further, Figure 3 shows that gains only occur when $P_{\text{replay}}$ considers *both* level scores and staleness ($0 < \rho < 1$), highlighting the importance of staleness-based sampling in keeping scores from drifting off-policy. Lastly, we also

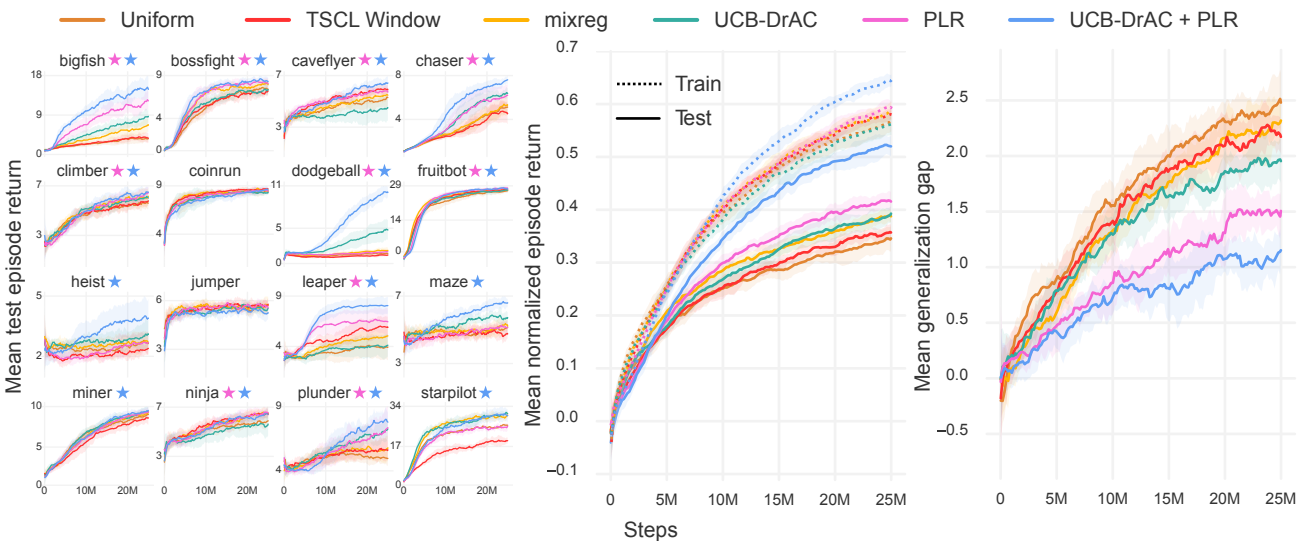


*Figure 2.* Left: Mean episodic test returns (10 runs) of each method. Each colored ★ indicates statistically significant ($p < 0.05$) gains in final test performance or sample complexity along the curve, relative to uniform sampling, for the PLR-based method of the same color. Center: Mean normalized train and test returns averaged across all games. Right: Mean generalization gaps averaged across all games.

benchmarked PLR on the hard setting against the same set of methods, where it again leads with 35% greater test returns relative to uniform sampling and 83% greater test returns when combined with UCB-DrAC. Figures 10–18 and Table 4 in Appendix B report additional details on these results.

The alternative scoring metrics based on TD-error and classifier uncertainty perform inconsistently across games. While certain games, such as BigFish, see improved sample-efficiency and generalization under various scoring functions, others, such as Ninja, see no improvement or worse, degraded performance. See Figure 3 for an example of this inconsistent effect across games. We find the best-performing variant of TSCL Window does not incorporate staleness information ($\rho = 0$) and similarly leads to inconsistent outcomes across games at test time, notably significantly worsening performance on StarPilot, as seen in Figure 2, and increasing the generalization gap on some environments as revealed in Figure 15 of Appendix B.

### 5.2. MiniGrid

We provide empirical support for the Value Correction Hypothesis (defined in Section 3) on two challenging MiniGrid environments, whose levels fall into discrete difficulties (e.g. by number of rooms to be traversed). In both, PLR with rank prioritization significantly improves sample efficiency and generalization over uniform sampling, demonstrating our method also works well in discrete state spaces. We find a staleness coefficient of $\rho = 0.3$ leads to the best test performance on MiniGrid. The top row of Figure 4 summarizes these results.

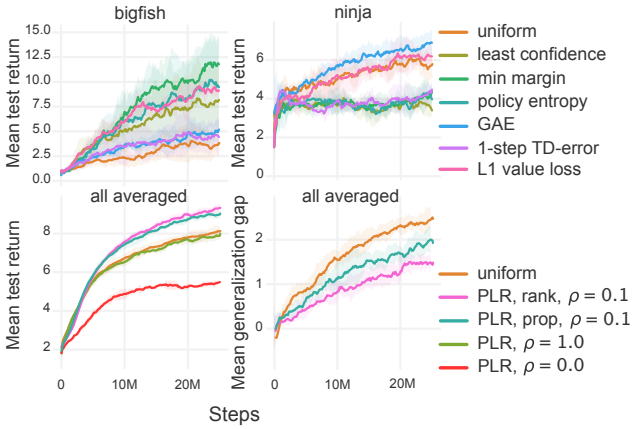


*Figure 3.* Top: Two example Procgen environments, between which all scoring functions except L1 value loss show inconsistent improvements to test performance (rank prioritization, $\beta = 0.1$, $\rho = 0.3$). This inconsistency holds across settings in our grid search. Bottom: Mean unnormalized episodic test returns (left) and mean generalization gap (right) for various PLR settings.

To test our hypothesis, we bin each level into its corresponding difficulty, expressed as ascending, discrete values (note that PLR does not have access to this privileged information). In the bottom row of Figure 4, we see how the expected difficulty of levels sampled using PLR changes during training for each environment. We observe that as $P_{\text{replay}}$ is updated, levels become sampled according to an implicit curriculum over the training levels that prioritizes progressively harder levels. Of particular note, PLR seems to struggle to discover a useful curriculum for around the first 4,000 updates on ObstructedMazeGamut-Medium, at which point it discovers

*Table 2.* Test returns of policies trained using each method with its best hyperparameters. Following Raileanu et al. (2021), the reported mean and standard deviations per environment are computed by evaluating the final policy's average return on 100 test episodes, aggregated across multiple training runs (10 runs for Procgen Benchmark and 3 for MiniGrid, each initialized with a different training seed). Normalized test returns per run are computed by dividing the average test return per run for each environment by the corresponding average test return of the uniform-sampling baseline over all runs. We then report the means and standard deviations of normalized test returns aggregated across runs. We report the normalized return statistics for Procgen and MiniGrid environments separately. Bolded methods are not significantly different from the method with highest mean, unless all are, in which case none are bolded.

| Environment | Uniform | TSCL | mixreg | UCB-DrAC | PLR | UCB-DrAC + PLR |
|---|---|---|---|---|---|---|
| BigFish | $3.7 \pm 1.2$ | $4.3 \pm 1.3$ | $6.9 \pm 1.6$ | $8.7 \pm 1.1$ | $10.9 \pm 2.8$ | $\mathbf{14.3 \pm 2.1}$ |
| BossFight | $7.7 \pm 0.4$ | $7.4 \pm 0.8$ | $8.1 \pm 0.7$ | $7.7 \pm 0.7$ | $\mathbf{8.9 \pm 0.4}$ | $\mathbf{8.8 \pm 0.8}$ |
| CaveFlyer | $5.4 \pm 0.8$ | $\mathbf{6.3 \pm 0.6}$ | $6.0 \pm 0.6$ | $4.6 \pm 0.9$ | $\mathbf{6.3 \pm 0.5}$ | $\mathbf{6.8 \pm 0.7}$ |
| Chaser | $5.2 \pm 0.7$ | $4.9 \pm 1.0$ | $5.7 \pm 1.1$ | $6.8 \pm 0.9$ | $6.9 \pm 1.2$ | $\mathbf{8.0 \pm 0.6}$ |
| Climber | $5.9 \pm 0.6$ | $6.0 \pm 0.8$ | $\mathbf{6.6 \pm 0.7}$ | $\mathbf{6.4 \pm 0.9}$ | $\mathbf{6.3 \pm 0.8}$ | $\mathbf{6.8 \pm 0.7}$ |
| CoinRun | $8.6 \pm 0.4$ | $\mathbf{9.2 \pm 0.2}$ | $8.6 \pm 0.3$ | $8.6 \pm 0.4$ | $8.8 \pm 0.5$ | $\mathbf{9.0 \pm 0.4}$ |
| Dodgeball | $1.7 \pm 0.2$ | $1.2 \pm 0.4$ | $1.8 \pm 0.4$ | $5.1 \pm 1.6$ | $1.8 \pm 0.5$ | $\mathbf{10.3 \pm 1.4}$ |
| FruitBot | $27.3 \pm 0.9$ | $27.1 \pm 1.6$ | $27.7 \pm 0.8$ | $27.0 \pm 1.3$ | $28.0 \pm 1.3$ | $27.6 \pm 1.5$ |
| Heist | $2.8 \pm 0.9$ | $2.5 \pm 0.6$ | $2.7 \pm 0.4$ | $3.2 \pm 0.7$ | $2.9 \pm 0.5$ | $\mathbf{4.9 \pm 1.3}$ |
| Jumper | $5.7 \pm 0.4$ | $6.1 \pm 0.6$ | $6.1 \pm 0.3$ | $5.6 \pm 0.5$ | $5.8 \pm 0.5$ | $5.9 \pm 0.3$ |
| Leaper | $4.2 \pm 1.3$ | $6.4 \pm 1.2$ | $5.2 \pm 1.1$ | $4.4 \pm 1.4$ | $6.8 \pm 1.2$ | $\mathbf{8.7 \pm 1.0}$ |
| Maze | $5.5 \pm 0.4$ | $5.0 \pm 0.3$ | $5.4 \pm 0.5$ | $6.2 \pm 0.5$ | $5.5 \pm 0.8$ | $\mathbf{7.2 \pm 0.8}$ |
| Miner | $8.7 \pm 0.7$ | $8.9 \pm 0.6$ | $9.5 \pm 0.4$ | $\mathbf{10.1 \pm 0.6}$ | $\mathbf{9.6 \pm 0.6}$ | $\mathbf{10.0 \pm 0.5}$ |
| Ninja | $6.0 \pm 0.4$ | $\mathbf{6.8 \pm 0.5}$ | $\mathbf{6.9 \pm 0.5}$ | $5.8 \pm 0.8$ | $\mathbf{7.2 \pm 0.4}$ | $\mathbf{7.0 \pm 0.5}$ |
| Plunder | $5.1 \pm 0.6$ | $5.9 \pm 1.1$ | $5.7 \pm 0.5$ | $\mathbf{7.8 \pm 0.9}$ | $\mathbf{8.7 \pm 2.2}$ | $\mathbf{7.7 \pm 0.9}$ |
| StarPilot | $26.8 \pm 1.5$ | $19.8 \pm 3.4$ | $\mathbf{32.7 \pm 1.5}$ | $\mathbf{31.7 \pm 2.4}$ | $27.9 \pm 4.4$ | $29.6 \pm 2.2$ |
| Normalized test returns (%) | $100.0 \pm 4.5$ | $103.0 \pm 3.6$ | $113.8 \pm 2.8$ | $129.8 \pm 8.2$ | $128.3 \pm 5.8$ | $\mathbf{176.4 \pm 6.1}$ |
| MultiRoom-N4-Random | $0.80 \pm 0.04$ | – | – | – | $\mathbf{0.81 \pm 0.01}$ | – |
| ObstructedMazeGamut-Easy | $0.53 \pm 0.04$ | – | – | – | $\mathbf{0.85 \pm 0.04}$ | – |
| ObstructedMazeGamut-Med | $0.65 \pm 0.01$ | – | – | – | $\mathbf{0.73 \pm 0.07}$ | – |
| Normalized test returns (%) | $100.0 \pm 2.5$ | – | – | – | $\mathbf{124.3 \pm 4.7}$ | – |

a curriculum that gradually assigns more weight to harder levels. This curriculum enables PLR with access to only $6{,}000$ training levels to attain even higher mean test returns than the uniform-sampling baseline with access to the full set of training levels, of which there are roughly $4$ billion (so our training levels constitute $0.00015\%$ of the total number).

We further tested an extended version of PLR that trains on the full level distribution on these two environments by tracking a buffer of levels with the highest estimated learning potential. We find it outperforms uniform sampling with access to the full level distribution. These additional results are presented in Appendix B.3.

## 6. Related Work

Several methods for improving generalization in deep RL adapt techniques from supervised learning, including stochastic regularization (Igl et al., 2019; Cobbe et al., 2020a), data augmentation (Kostrikov et al., 2020; Raileanu et al., 2021; Wang et al., 2020a), and feature distillation (Igl et al., 2020; Cobbe et al., 2020b). In contrast, PLR modifies only how the next training level is sampled, thereby easily combining with any model or RL algorithm.

The selective-sampling performed by PLR makes it a form of active learning (Cohn et al., 1994; Settles, 2009). Our work also echoes ideas from Graves et al. (2017), who train a multi-armed bandit to choose the next task in multi-task supervised learning, so to maximize gradient-based progress signals. Sharma et al. (2018) extend these ideas to multi-task RL, but add the additional requirement of knowing a maximum target return for each task a priori. Zhang et al. (2020b) use an ensemble of value functions for selective goal sampling in the off-policy continuous control setting, requiring prior knowledge of the environment structure to generate candidate goals. Unlike PLR, these methods assume the ability to sample tasks or levels based on their structural properties, an assumption that does not typically hold for PCG simulators. Instead, our method automatically uncovers similarly difficult levels, giving rise to a curriculum without prior knowledge of the environment.

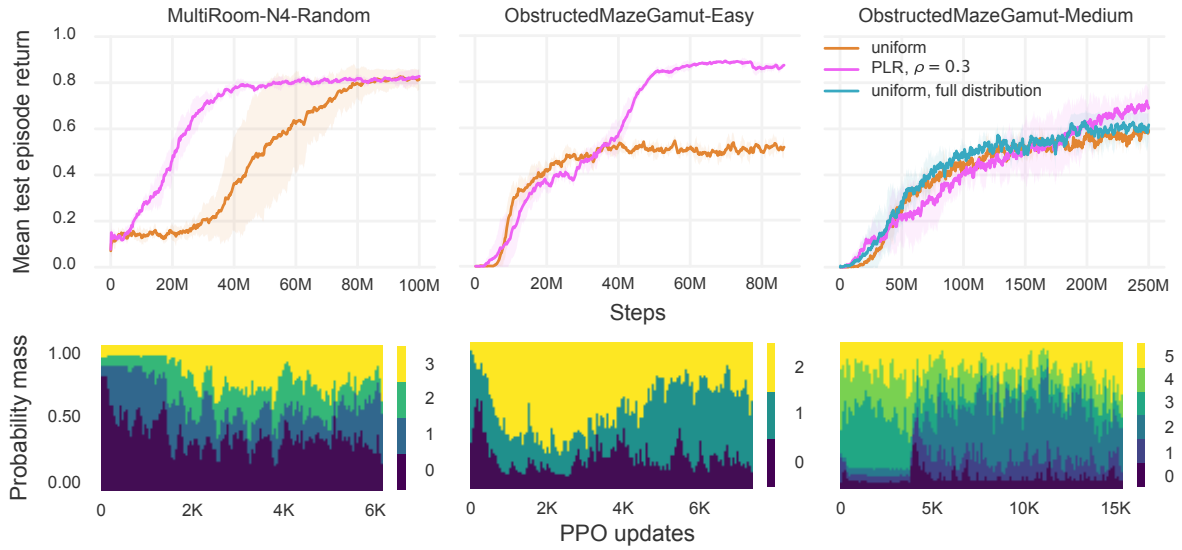


*Figure 4.* Top: Mean episodic test returns of PLR and the uniform-sampling baseline on MultiRoom-N4-Random (4 runs), ObstructedMazeGamut-Easy (3 runs), and ObstructedMazeGamut-Medium (3 runs). Bottom: The probability mass assigned to levels of varying difficulty over the course of training in a single, randomly selected run for the respective environment.

A recent theme in the PCG setting explores adaptively generating levels to facilitate learning (Sukhbaatar et al., 2017; Wang et al., 2019; 2020b; Khalifa et al., 2020; Akkaya et al., 2019; Campero et al., 2020; Dennis et al., 2020). Unlike these approaches, our method does not assume control over level generation, requiring only the ability to replay previously visited levels. These methods also require parameterizing level generation with additional learning modules. In contrast, our approach does not require such extensions of the environment, for example including teacher-specific action spaces in the case of Campero et al. (2020). Most similar to our method, Matiisen et al. (2020) proposes a teacher-student curriculum learning (TSCL) algorithm that samples training levels by considering the change in episodic returns per level, though they neither design nor test the method for generalization. As shown in Section 5.1, it provides inconsistent benefits at test time. Further, unlike TSCL, PLR does not assume access to all levels at the start of training, and as we show in Appendix B.3, PLR can be extended to improve sample-efficiency and generalization by training on an unbounded number of training levels.

Like our method, Schaul et al. (2016) and Kapturowski et al. (2019) use TD-errors to estimate learning potential. While these methods make use of TD-errors to prioritize learning from *past* experiences, our method uses such estimates to prioritize revisiting levels for generating entirely new *future* experiences for learning.

Generalization requires sufficient exploration of environment states and dynamics. Thus, recent exploration strategies (e.g. Raileanu & Rocktäschel, 2020; Campero et al., 2020; Zhang et al., 2020a; Zha et al., 2021) shown to benefit simple PCG settings are complementary to the aims of this work. However, as these studies focus on PCG environments with low-dimensional state spaces, whether such methods can be successfully applied to more complex PCG environments like Procgen Benchmark remains to be seen. If so, they may potentially combine with PLR to yield additive improvements. We believe the interplay between such exploration methods and PLR to be a promising direction for future research.

## 7. Conclusion and Future Work

We introduced Prioritized Level Replay (PLR), an algorithm for selectively sampling the next training level in PCG environments based on the estimated learning potential of revisiting each level for the current policy. We showed that our method remarkably improves both the sample-efficiency and generalization of deep RL agents in PCG environments, including the majority of environments in Procgen Benchmark and two challenging MiniGrid environments. We further combined PLR with the prior leading method to set a new state-of-the-art on Procgen Benchmark. Further, on MiniGrid environments, we showed PLR induces an emergent curriculum of increasingly more difficult levels.

The flexibility of the PCG abstraction makes PLR applicable to many problems of practical importance, for example, robotic object manipulation tasks, where domain randomized environment instances map to the notion of levels. We believe PLR may even be applicable to singleton environments, given a procedure for generating variations of the underlying MDP as a function of a level identifier, for ex-

ample, by varying the starting positions of entities. Another natural extension of PLR is to adapt the method to operate in the goal-conditioned setting, by incorporating goals into the level parameterization.

Despite the wide applicability of PCG and consequently PLR, not all problem domains can be effectively represented in seed-based simulation. Many real world problems require transfer into domains too complex to be adequately captured by simulation, such as car driving, where realizing a completely faithful simulation would entail solving the very same control problem of interest, creating a chicken-and-egg dilemma. Further, environment resets are not universally available, such as in the continual learning setting, where the agent interacts with the environment without explicit episode boundaries—arguably, a more realistic interaction model for a learning agent deployed in the wild.

Still, pre-training in simulation with resets can nevertheless benefit such settings, where the target domain is rife with open-ended complexity and where resets are unavailable, especially as training through real-world interactions can be slow, expensive, and precarious. In fact, in practice, we almost exclusively train deep RL policies in simulation for these reasons. As PLR provides a simple method to more fully exploit the simulator for improved test-time performance, we believe PLR can also be adapted to improve learning in these settings.

We further note that while we empirically demonstrated that the L1 value loss acts as a highly effective scoring function, there likely exist even more potent choices. Directly learning such functions may reveal even better alternatives. Lastly, combining PLR with various exploration strategies may further improve test performance in hard exploration environments. We look forward to investigating each of these promising directions in future work, prioritized accordingly, by learning potential.

## Acknowledgements

We thank Roberta Raileanu, Heinrich Küttler, and Jakob Foerster for useful discussions and feedback on this work, and our anonymous reviewers, for their recommendations on improving this paper.

## References

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., and Bowling, M. The hanabi challenge: A new frontier for AI research. *Artif. Intell.*, 280:103216, 2020. doi: 10.1016/j.artint.2019.103216. URL https://doi.org/10.1016/j.artint.2019.103216.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Campero, A., Raileanu, R., Küttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E. Learning with AMIGo: Adversarially Motivated Intrinsic Goals. *CoRR*, abs/2006.12122, 2020. URL https://arxiv.org/abs/2006.12122.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. BabyAI: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272, 2018. URL http://arxiv.org/abs/1810.08272.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid, 2018.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *International Conference on Machine Learning*, pp. 2048–2056. PMLR, November 2020a. URL http://proceedings.mlr.press/v119/cobbe20a.html. ISSN: 2640-3498.

Cobbe, K., Hilton, J., Klimov, O., and Schulman, J. Phasic Policy Gradient. *CoRR*, abs/2009.04416, 2020b. URL https://arxiv.org/abs/2009.04416.

Cohn, D., Atlas, L., and Ladner, R. Improving generalization with active learning. *Machine learning*, 15(2): 201–221, 1994.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A. M., Russell, S., Critch, A., and Levine, S. Emergent complexity and zero-shot transfer via unsupervised environment design. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1311–1320. PMLR, 2017. URL http://proceedings.mlr.press/v70/graves17a.html.

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., and Hofmann, K. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pp. 13978–13990, 2019.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. The impact of non-stationarity on generalisation in deep reinforcement learning. *CoRR*, abs/2006.05826, 2020. URL https://arxiv.org/abs/2006.05826.

Juliani, A., Khalifa, A., Berges, V.-P., Harper, J., Teng, E., Henry, H., Crespi, A., Togelius, J., and Lange, D. Obstacle tower: A generalization challenge in vision, control, and planning. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Aug 2019. doi: 10.24963/ijcai.2019/373. URL http://dx.doi.org/10.24963/ijcai.2019/373.

Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729, 2018. URL http://arxiv.org/abs/1806.10729.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=r1lyTjAqYX.

Khalifa, A., Bontrager, P., Earle, S., and Togelius, J. PCGRL: procedural content generation via reinforcement learning. *CoRR*, abs/2001.09212, 2020. URL https://arxiv.org/abs/2001.09212.

Kostrikov, I., Yarats, D., and Fergus, R. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. *CoRR*, abs/2004.13649, 2020. URL https://arxiv.org/abs/2004.13649.

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. The nethack learning environment. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/569ff987c643b4bedf504efda8f786c2-Abstract.html.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31 (9):3732–3740, Sep 2020. ISSN 2162-2388. doi: 10.1109/tnnls.2019.2934906. URL http://dx.doi.org/10.1109/TNNLS.2019.2934906.

Raileanu, R. and Rocktäschel, T. RIDE: rewarding impact-driven exploration for procedurally-generated environments. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkg-TJBFPB.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in reinforcement learning, 2021. URL https://openreview.net/forum?id=9l9WD4ahJgs.

Risi, S. and Togelius, J. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, Aug 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0208-z. URL http://dx.doi.org/10.1038/s42256-020-0208-z.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05952.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using

generalized advantage estimation. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Settles, B. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Sharma, S., Jha, A. K., Hegde, P., and Ravindran, B. Learning to multi-task by active sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=B1nZ1weCZ.

Sukhbaatar, S., Kostrikov, I., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. *CoRR*, abs/1703.05407, 2017. URL http://arxiv.org/abs/1703.05407.

Wang, K., Kang, B., Shao, J., and Feng, J. Improving generalization in reinforcement learning with mixture regularization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020a. URL https://proceedings.neurips.cc/paper/2020/hash/5a751d6a0b6ef05cfe51b86e5d1458e6-Abstract.html.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. POET: open-ended coevolution of environments and their optimized solutions. In Auger, A. and Stützle, T. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pp. 142–151. ACM, 2019. doi: 10.1145/3321707.3321799. URL https://doi.org/10.1145/3321707.3321799.

Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., and Stanley, K. Enhanced POET: Open-ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions. In *International Conference on Machine Learning*, pp. 9940–9951. PMLR, November 2020b. URL http://proceedings.mlr.press/v119/wang20l.html. ISSN: 2640-3498.

Zha, D., Ma, W., Yuan, L., Hu, X., and Liu, J. Rank the episodes: A simple approach for exploration in procedurally-generated environments. *CoRR*, abs/2101.08152, 2021. URL https://arxiv.org/abs/2101.08152.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., and Tian, Y. Bebold: Exploration beyond the boundary of explored regions. *CoRR*, abs/2012.08621, 2020a. URL https://arxiv.org/abs/2012.08621.

Zhang, Y., Abbeel, P., and Pinto, L. Automatic curriculum learning through value disagreement. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020b. URL https://proceedings.neurips.cc/paper/2020/hash/566f0ea4f6c2e947f36795c8f58ba901-Abstract.html.

Zhong, V., Rocktäschel, T., and Grefenstette, E. RTFM: generalising to new environment dynamics via reading. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgob6NKvH.