On Unsupervised Environment Design

Nathan Monette

September 2025

1 Introduction

These are a set of notes that I've compiled in order to provide a resource for those interested in the area of Unsupervised Environment Design (Dennis et al., 2020, UED). I'll try to be somewhat precise mathematically, but hopefully not to the extent that the ideas are inaccessible.

For starters, UED is a reinforcement learning (RL) *framework* for training a generalist agent. In particular, the axis of generalisation is across *levels* of an environment, meaning different configurations of the same environment. A popular example (and benchmark) of such a configurable environment is the Minigrid Maze (Chevalier-Boisvert et al., 2023), an implementation of which is shown in Figure 1.

Figure 1: A screenshot of different Minigrid maze levels from the JaxUED library (Coward et al., 2024).

UED is framed as a *game*, where at each training iteration, there is an adversary that plays a distribution over training levels, and the agent trains on levels sampled from that distribution. In the robustness sense, the adversary is supposed to propose levels that challenge the agent. The intention here (informally) is that at some point in this game where neither player can improve their strategies, the agent can generalise across the level set. However, for the sake of efficiency, we are also concerned with providing the agent with a *curriculum*, which transitions the agent gradually from easier levels to harder levels throughout training in

order to guide its learning process. Thus, the adversarial training mechanism of UED in some sense is a form of an *autocurriculum*.

2 UED Basics

2.1 Notation

In terms of notation, we will mostly use that of Monette et al. (2025). UED is formalised by the Underspecified POMDP, which defines some POMDP $(\mathcal{S}, \mathcal{O}, \mathcal{A}, r, P, \rho, \gamma)$ for each level λ in level set $\mathcal{L}$. $\mathcal{S}$ is the state space, $\mathcal{O}$ is the observation space, $\mathcal{A}$ is the action space, and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{L} \to \mathbb{R}$ is the reward function. Different from Monette et al. (2025), we define the function $\mathcal{D}(\cdot)$ as the function that takes any set to the set of probability distributions over that set. Thus, we define the transition distribution function $P : \mathcal{S} \times \mathcal{A} \times \mathcal{L} \to \mathcal{D}(\mathcal{S})$, the initial state distribution function $\rho : \mathcal{L} \to \mathcal{D}(\mathcal{S})$.

Moreover, we write the agent's neural policy as $\pi_x : \mathcal{O} \to \mathcal{D}(\mathcal{A})$, where $x \in \mathcal{X}$ is the parameters of the policy. Moreover, we write y as the adversary's parameters, $\Lambda \subseteq \mathcal{L}$ as the set of levels the adversary has access to, and $\Lambda(y)$ as the sampling distribution over the level set parameterised by y. $J(\pi_x, \lambda)$ is the policy's expected return on level λ, and $J(\pi_x, \Lambda)$ is the vector of policy returns on the levels in Λ. Similarly, we describe the policy's *score* as $s(\pi_x, \lambda)$ and $s(\pi_x, \Lambda)$, which is some function of the policy that the adversary uses to optimise its distribution. Finally, we describe the policy's trajectory distribution on a level to be $\pi_x(\lambda)$.

2.2 General UED Algorithm

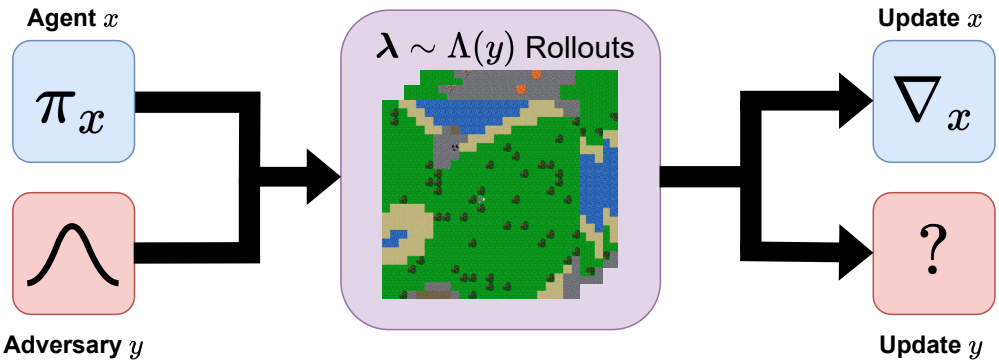

Figure 2: A visualisation of the general UED procedure. The policy trains on sampled levels from the adversary, and then the policy and (sometimes) the adversary are updated.

In general, the UED training regime takes the following steps at each iteration (roughly in order):

1. Sample levels $\boldsymbol{\lambda} \sim \Lambda(y)$.
2. Sample trajectories $\boldsymbol{\tau} \sim \pi_x(\boldsymbol{\lambda})$.

3. Update the agent's parameters x using reinforcement learning.
4. Update the adversary's parameters y using a variety of techniques (e.g. gradient optimisation, heuristics, evolution, or keeping it fixed)

Practically speaking, UED as a research area mostly concerns itself with steps **1** and **4**, and can use essentially any RL algorithm to train its underlying agent (at your own risk, because in practice pretty much all of the UED papers use PPO). These steps are summarised in Figure 2.

2.3 General Assumptions of UED

Firstly, there are some ideas that set the framework for how UED can be run. Primarily, we *do not* have access to the test distribution of levels, however we usually have access to the full set of levels $\mathcal{L}$ (usually exponentially large or even continuous), or at least some very large subset. Moreover, in order for UED methods to create effective curricula, there generally has to be levels of varying difficulties.

3 Warm-up: Uniform Randomisation

3.1 Domain Randomisation

Domain Randomisation (Tobin et al., 2017, DR) is the simplest UED algorithm, to the extent that it may be considered a "degenerate case" of other methods (Jiang et al., 2021a). DR is really simple: it samples levels uniformly over $\mathcal{L}$, with a fixed level set and distribution. In practice, we often do DR over some subset of $\mathcal{L}$, but this may lead to some bias in the training process. Regardless, as per Coward et al. (2024), more complicated methods may not actually outperform DR in some settings. According to the results from Coward et al. (2024), my hypothesis about this is that when the levels are not particularly expressive, good levels are very uncommon, or for some other reason, there must be a very high throughput of levels, then it may be difficult to outperform DR without doing a very intensive search/filtering step.

3.2 Sampling For Learnability

Say that we *do* want to perform filtering on the level space. What would be a good procedure for doing this? One answer is Sampling For Learnability (Rutherford et al., 2024, SFL), which performs a large filtering step every N training iterations, and then samples levels uniformly from the top-k scoring levels from the filtering step over the next N iterations. One assumption made by SFL is that it assumes a deterministic environment, because scoring a level where we have high aleatoric uncertainty is a difficult problem to solve in practice, and while there are proposed solutions, they don't really scale (Jiang et al., 2022).

In order to perform this filtering step, many levels are sampled, and the policy is evaluated for a number of episodes on each level. From the trajectories on each level, we then evaluate

the *variance of outcomes* on each level, called *learnability*. If the environment has binary outcomes, we can write this with the policy's solve rate on level λ, p_λ:

$$s(\pi_x, \lambda) = p_\lambda \cdot (1 - p_\lambda). \tag{1}$$

In other words, this is the variance of the Bernoulli distribution describing the policy's outcome. In the more general (still deterministic) case, the *generalised learnability* was written as the variance of returns scaled with a Gaussian fit to the level scores in Monette et al. (2025), but I'll leave the details to that paper, since they are a bit confusing.

4 Generator-Based Methods

4.1 PAIRED

Alongside the formal introduction of UED as a framework, Dennis et al. (2020) introduced PAIRED, which used what I like to call a "RL adversary." In particular, the way that levels are sampled in PAIRED is by having the adversary play an RL environment that involves designing the levels in some way. In the maze example, this might involve placing the walls, goal, and agent in their respective grid cells. Some contemporary methods (Garcin et al., 2024) have also replaced the RL adversary with generative models like a VAE to some success.

Moreover, PAIRED actually trains three total agents. The adversary is trained to maximise the regret of the agent, which would typically be written in terms of the optimal policy π_*:

$$s(\pi_x, \lambda) = J(\pi_*, \lambda) - J(\pi_x, \lambda). \tag{2}$$

However, as π_* is typically not available, PAIRED trains two RL agents: the protagonist (which we would call the agent) and also the antagonist. The agents are trained with RL on the same levels as each other (batches sampled from the adversary), however, if we write the protagonist as π_P and antagonist as π_A, the adversary gets the score

$$s(\pi_x, \lambda) = J(\pi_A, \lambda) - J(\pi_P, \lambda) \tag{3}$$

which serves as the (sparse) reward for the end of each episode of its RL level generation. Notably, the antagonist is differentiated from the protagonist via the *coordination assumption*, meaning that there is some form of coordination between the adversary and antagonist.

PAIRED tends to not do so well in comparison to more modern methods (Parker-Holder et al., 2022; Coward et al., 2024), but it does have the advantage of being a zero-sum game, so some modification of it could possibly have interesting convergence properties.

4.2 ADD

An improved approach for generating levels is to use diffusion (Chung et al., 2024, ADD). In ADD, image representations of each level are generated, and those images are parsed to get parameters that are usable by the underlying RL environment. ADD also uses *classifier*

guidance with the diffusion model, where an approximate regret bin classification model is used to "steer" the diffusion process towards a level that would cause agent to suffer high regret. ADD has the benefit of using diffusion (which is powerful), but the unfortunate side effect of using images as its level representation, which seems unwieldy.

5 Replay-Based Methods

5.1 PLR

Prioritised Level Replay (Jiang et al., 2021b,a, PLR) led to the result that the *curation* of a dynamic buffer of levels Λ to re-play might be a more effective way to perform UED. It is more efficient because it does not require the adversary to generate new levels (and requires minimal overhead for the adversary to adjust its parameters), but also seems to perform better than PAIRED.

PLR operates between two "modes:" sampling new levels and replaying old levels. In implementation (see: Jiang et al. (2023); Coward et al. (2024)) this means sampling a Bernoulli random variable at the beginning of each training iteration, where one outcome makes $\Lambda(y)$ the uniform distribution over $\mathcal{L}$, and the other outcome makes $\Lambda(y)$ the distribution over the levels in Λ, parameterised by the normalised scores of the levels in the buffer (with the addition of some heuristics that are something to the effect of entropy regularisation).

Notably, these heuristics may be prohibitive of a convergent adversary (e.g. the rank prioritisation: see Jiang et al. (2021b)). In order to compute $\Lambda(y)$, the buffer must maintain a score for each level from the last time it was sampled. Thus, PLR typically relies on a score function that can be computed from just training trajectories, such as the value loss of the critic averaged across the episode.[1] Also, when sampling uniformly over $\mathcal{L}$, high-scoring levels are then inserted into the buffer Λ in place of low scoring levels.

5.2 ACCEL

A more performant version of PLR is called ACCEL (Parker-Holder et al., 2022), which adds in an additional step on replay mode: *level mutation*. This requires the environment to have a *mutation operator* in order to edit the levels, but this enables the adversary to engage in *evolution*. Essentially, after performing an agentic update in replay mode, the agent is then rolled out on a batch of edited levels, and the levels where the policy receives a higher score than a level in the buffer, the edited level will be inserted into the buffer.

5.3 NCC

In PLR$^{\perp}$ (Jiang et al., 2021a), the uniform mode of sampling is effectively an *evaluation step*. On the principle that evaluation is cheap in comparison to training, we can reasonably evaluate on more levels than we actually train on. NCC (Monette et al., 2025) takes this principle to the extreme, and never actually keeps a buffer of scores, but rather evaluates the

[1] This score function does not perform that well but I'm using it for sake of example.

policy on the entire level buffer Λ *every* training iteration. Practically speaking it doesn't have to change every iteration, although that is how the paper was carried out.

NCC was the first UED paper that had provable convergence[2], and it does this in a very similar way to PLR, except it uses projected gradient ascent on the following objective to optimise the adversary:

$$\begin{aligned}\max_{y \in \mathcal{Y}} \; & f(x, y) \\ f(x, y) &= \mathbb{E}_{\lambda \sim \Lambda(y)}\Big[s(\pi_x, \lambda)\Big] + \alpha \mathcal{H}(y) \\ &= y^T s(\pi_x, \Lambda) + \alpha \mathcal{H}(y)\end{aligned}$$

where $\alpha > 0$ is the regularisation coefficient for entropy $\mathcal{H}(\cdot)$. Notably, this objective is strongly concave, so the optimisation of y is theoretically easy. Some intuition about this objective is that the adversary wants to find the levels which maximise the score, while maintaining some diversity (entropy) over the level set.

According to Monette et al. (2025), NCC performs quite well, however it requires large amounts of evaluation which may be infeasible in some environments (particularly ones not in JAX). In particular, when it has access to true regret, it seems to produce policies that are robust, as its theoretical version's guarantees point to.

6 Conclusion

Thanks for checking this out! Please don't hesitate to reach out if you have any questions or feedback :).

References

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 73383–73394. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/e8916198466e8ef218a2185a491b49fa-Paper-Datasets_and_Benchmarks.pdf.

Hojun Chung, Junseo Lee, Minsoo Kim, Dohyeong Kim, and Songhwai Oh. Adversarial environment design via regret-guided diffusion models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=eezCLKwx6T.

[2]Although the experiments in the paper use a practical version of the method that lacks guarantees

Samuel Coward, Michael Beukman, and Jakob Foerster. Jaxued: A simple and useable ued library in jax. *arXiv preprint*, 2024.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *NeurIPS*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Samuel Garcin, James Doran, Shangmin Guo, Christopher G. Lucas, and Stefano V. Albrecht. Dred: Zero-shot transfer in reinforcement learning via data-regularised environment design. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Minqi Jiang, Michael D Dennis, Jack Parker-Holder, Jakob Nicolaus Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021a. URL https://openreview.net/forum?id=5UZ-AcwFDKJ.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 18–24 Jul 2021b. URL https://proceedings.mlr.press/v139/jiang21b.html.

Minqi Jiang, Michael D Dennis, Jack Parker-Holder, Andrei Lupu, Heinrich Kuttler, Edward Grefenstette, Tim Rocktäschel, and Jakob Nicolaus Foerster. Grounding aleatoric uncertainty for unsupervised environment design. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=AbLjOl8YbYt.

Minqi Jiang, Michael Dennis, Edward Grefenstette, and Tim Rocktäschel. minimax: Efficient baselines for autocurricula in jax. 2023.

Nathan Monette, Alistair Letcher, Michael Beukman, Matthew Thomas Jackson, Alexander Rutherford, Alexander David Goldie, and Jakob Nicolaus Foerster. An optimisation framework for unsupervised environment design. In *Reinforcement Learning Conference*, 2025. URL https://openreview.net/forum?id=WnknYUybWX.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob N. Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *ICML*, pp. 17473–17498, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Nicolaus Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=iEeiZlTbts.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. IEEE Press, 2017. doi: 10.1109/IROS.2017.8202133. URL https://doi.org/10.1109/IROS.2017.8202133.