Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2025)

Video Game Level Design as a Multi-Agent Reinforcement Learning Problem

Sam Earle[1], **Zehua Jiang**[1], **Eugene Vinitsky**[1], **Julian Togelius**[1, 2]
[1]New York University
[2]University of Skövde
sam.earle@nyu.edu

Abstract

Procedural Content Generation via Reinforcement Learning (PCGRL) offers a method for training controllable level designer agents without the need for human datasets, using metrics that serve as proxies for level quality as rewards. Existing PCGRL research focuses on single generator agents, but are bottlenecked by the need to frequently recalculate heuristics of level quality and the agent's need to navigate around potentially large maps. By framing level generation as a multi-agent problem, we mitigate the efficiency bottleneck of single-agent PCGRL by reducing the number of reward calculations relative to the number of agent actions. We also find that multi-agent level generators are better able to generalize to out-of-distribution map shapes, which we argue is due to the generators' learning more local, modular design policies. We conclude that treating content generation as a distributed, multi-agent task is beneficial for generating functional artifacts at scale.

Code — https://github.com/smearle/pcgrl-jax

Introduction

Level design is a crucial part of video game development, and much work has gone into finding algorithmic means of generating or assisting in the design of video game levels. Generally, level generation is considered a form of Procedural Content Generation (PCG). This includes both level generation during gameplay, as is common in roguelikes and related game genres, and design assistance during game development. In addition to games for entertainment purposes, level generation is important for serious games (e.g. training simulations), as well as game-based AI benchmarks and training environments.

Many different methods have been developed for different forms of PCG, and in particular for level generation. In Procedural Content Generation via Reinforcement Learning (PCGRL, Khalifa et al. 2020), level-designing agents are trained to generate levels, and rewarded for producing designs that satisfy some reward function. This method offers fast generation times and a high degree of controllability, but requires long and costly training runs. This is because of the need for frequent, costly reward calculations, typically involving the computation of global shortest paths. Unlike most RL environment dynamics which are strictly local, these path-length computations have complexity $O(N^2)$, where N is the maximum width of a given map.

This paper investigates whether a multi-agent approach can improve results in PCGRL. Because PCGRL is a long-horizon problem involving rewards that depend on global, combinatorial features that may be hard to model, it is challenging to approach with RL techniques. Prior work has observed that decreasing the size of the observations that agents use, from observing the full map to solely observing a small local region, leads to empirical improvements in efficacy and generalization (Earle, Jiang, and Togelius 2024). This finding suggests that although maze-like level construction does involve global reasoning, much of the task can be accomplished locally. This in turn suggests that the problem can be factorized without much loss into sub-problems.

We hypothesize that decomposing PCGRL for game level construction into a decentralized problem, in which multiple agents seek to cooperatively build a level, may yield significant improvements in PCGRL capabilities. Agents can efficiently learn to construct levels for the local region they are tasked with, while approaching global coherence by exchanging information via overlapping regions that they jointly observe.

This type of decomposition, if possible, is critical for efficient PCGRL. By converting the single-agent PCGRL problem into a multi-agent one, we achieve several key benefits. First, the horizon of the problem is potentially reduced, with each agent taking fewer actions before the entire level has been edited by the collective. Second, for vectorized simulators where the cost of stepping agent dynamics is low, it is often as fast to simulate one agent as it is to simulate many. Instead, for PCGRL environments, the step-time of the environment is heavily dominated by the cost of computing rewards at each step. By sharing reward amongst the agents at each step, we can reduce the cost of stepping the most expensive part of the environment. Finally, the factorization of PCGRL tasks into local sub-problems may yield generalization benefits as has been observed in other work (Earle, Jiang, and Togelius 2024).

In sum, our contributions are as follows:

1. We frame video game level design as a multi-agent rein-

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

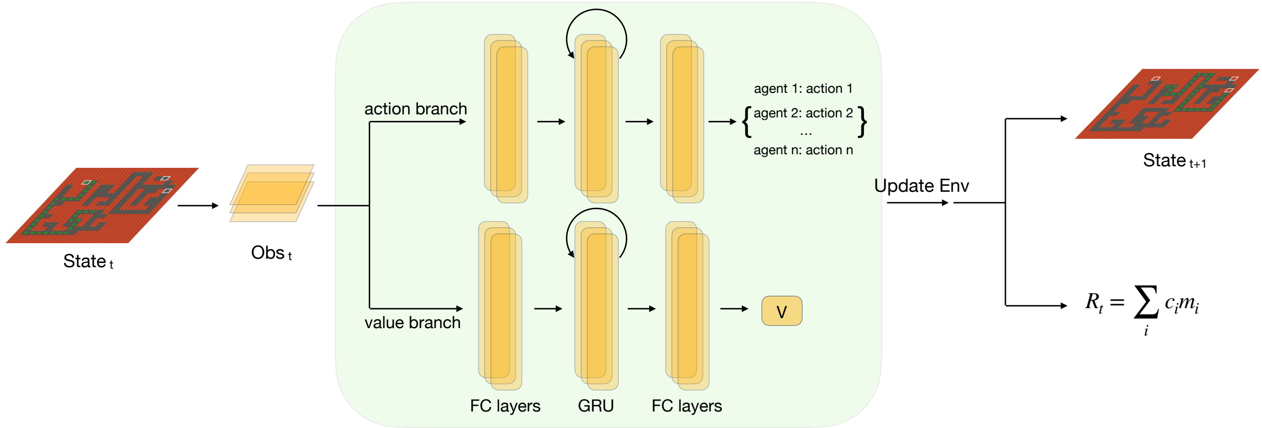

Figure 1: In multi-agent PCGRL, agent actions are taken in parallel, reducing the number of reward computations required relative to level edits, while maintaining a per-agent dense reward scheme, and fostering collaboration between agents. Reward R_t is computed according the weighted sum of heuristic scores m_i.

	mean episode reward							
map shape	fixed				random			
map width	8	16	24	32	8	16	24	32
n. agents								
1	18.09 ± 1.15	46.30 ± 3.71	107.92 ± 4.98	156.10 ± 6.73	6.38 ± 1.50	20.26 ± 3.04	35.50 ± 4.99	68.36 ± 7.90
2	19.66 ± 2.30	55.33 ± 3.86	113.94 ± 5.17	167.13 ± 5.77	7.25 ± 0.97	24.13 ± 3.72	40.57 ± 3.00	78.99 ± 5.97
3	**20.71 ± 2.60**	**62.57 ± 6.77**	**124.81 ± 6.68**	**181.81 ± 11.80**	**7.73 ± 1.37**	**25.30 ± 2.96**	**47.17 ± 3.57**	**88.43 ± 5.68**

Table 1: Adding agents—with a shared policy and shared reward—improves performance and generalization on a maze level generation task, both on (fixed, 16×16) maps seen during training, and on maps of different maximum sizes and/or with randomized rectangular shapes. Agents are allowed twice as many environment steps as there are tiles on the map. Values are averaged over models from 10 training run seeds, each evaluated over 50 episodes in each evaluation setting.

forcement learning problem.

2. We extend the PCGRL framework for level design via RL to support a multi-agent setting, implemented in JAX. The training loop and environment are fully parallelized and run entirely on the GPU, enabling rapid experimentation.
3. We investigate the effect on performance and generalization of the number of agents, number of episode steps, and reward-computation frequency, showing that additional agents provide performance, generalization and efficiency gains relative to single-agent PCGRL.

Related Work

Procedural Content Generation via Reinforcement Learning

Procedural Content Generation via Machine Learning (PCGML, Summerville et al. 2018) is an active field of research that explores the capacity for AI systems to generate game content. Much research has been conducted in supervised and self-supervised PCGML methods for level generation. Assuming a dataset of game levels, models can be trained using supervised learning to generate similar game levels (Snodgrass and Ontanón 2016; Summerville and Mateas 2016; Merino et al. 2023). Self-supervised methods such as Generative Adversarial Networks and Variational Autoencoders have also exhibited success in level generation (Torrado et al. 2019; Awiszus, Schubert, and Rosenhahn 2020; Sarkar and Cooper 2021). Recent research has even explored the capacity of Large Language Models as procedural content generators (Todd et al. 2023; Sudhakaran et al. 2023; Merino et al. 2024; Earle, Parajuli, and Banburski-Fahey 2025).

Procedural Content Generation via Reinforcement Learning (PCGRL) is a subset of PCGML that prioritizes the functional characteristics of generated levels over their aesthetics. In a sense, it represents an inversion of the typical RL paradigm: rather than training an agent to *play* game levels, PCGRL trains an agent to *create* them. To this end, level design is itself cast as a kind of game—an iterative process in which agents make local edits in order to maximize functional metrics over a long time horizon. Single-agent PCGRL was first explored by Khalifa et al. (2020), where they applied the method to generate 2D tile-based binary mazes, Zelda-like dungeons and Sokoban levels.

In PCGRL, level generation is modeled as a sequential decision making process. At each step, the agent receives an observation, a reward, and selects an action. Unlike tradi-

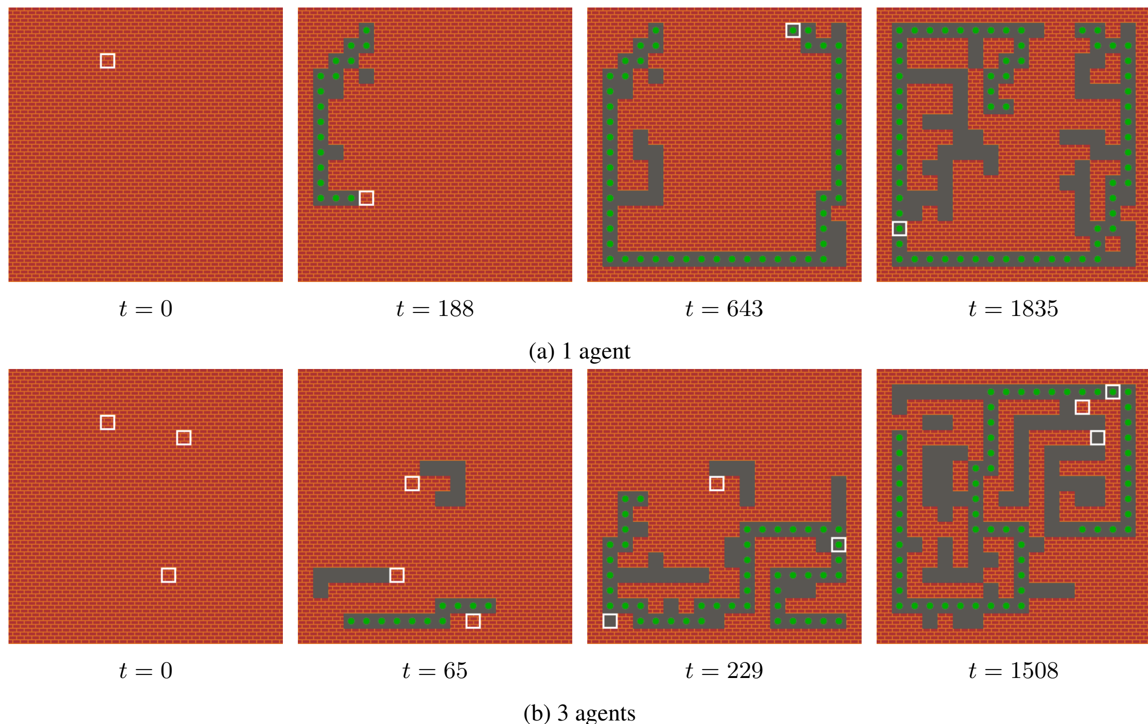

Figure 2: Episode rollouts with variable numbers of agents. Having multiple agents tends to allow for better map coverage and more complex and modular design patterns. For the sake of illustration, levels are initialized full of wall tiles. t represents the timestep of a given frame.

tional PCGML methods that generate entire levels in a single pass, PCGRL generates levels by modifying a single tile at a time. Each episode starts with an N^2 array of tiles randomly initialized from a predefined, discrete tile set. The agent changes a single tile per time-step and earns a reward based on how the change affects certain global functional characterstics of the level relative to some target values (e.g. a desired path-length or number of connected traversible regions). In follow-up work, the PCGRL framework was extended to support variable functional targets, with agents learning to produce diverse levels in response to conditional inputs Earle et al. (2021). Jiang et al. (2022) apply PCGRL to 3D mini-games in MineCraft, and (Earle, Jiang, and Togelius 2024) further probe the approach's scalability by training generators to work around "frozen" pivotal tiles and evaluating generalization on out-of-distribution map shapes.

Notably, PCGRL does not require any training data; computational costs are largely front-loaded into the training process. Once a generator has been trained, inference is cheap, a critical component in enabling real-time content generation. This stands in contrast to search-based PCG (Togelius et al. 2011), which has seen limited adoption in runtime content generation in games largely because of its high latency in combinatorially complex search spaces.

PCGRL has two major intended applications. One is to produce efficient generators for environments in which to train robust and generally-capable embodied agents. Along these lines, PCGRL has influenced the level generation approaches in Unsupervised Environment Design (Dennis et al. 2020; Jiang, Grefenstette, and Rocktäschel 2021; Parker-Holder et al. 2022). The other application is to produce effective generators for use in co-creative pipelines alongside human designers. Much prior work has examined the use of real-time level-generators as co-creative tools, using methods such as interactive evolution (Schrum et al. 2020), quality diversity evolutionary search (Charity, Khalifa, and Togelius 2020), and supervised learning (Guzdial et al. 2019). Delarosa et al. (2021) explore the use of PCGRL agents as design assistants with *RL Brush*, an online tool for Sokoban level design in which users can select between suggested edits from a diverse set of trained models, make manual edits, and playtest partial levels.

Multi-Agent Reinforcement Learning

Multi-agent RL has been applied both to real-world control problems, such as coordination of drone swarms (Hüttenrauch, Šošić, and Neumann 2017) and autonomous vehicle fleets (Shalev-Shwartz, Shammah, and Shashua 2016), and virtual ones, such as the control of multi-unit teams in the real-time strategy video game *StarCraft* (Ellis et al. 2023). While the objectives of agents in the above domains are shared, they may also be competitive or mixed (Buşoniu, Babuška, and De Schutter 2010). In Artificial Life-like simulations like NeuralMMO, for example, per-agent rewards lead to a variety of emergent and variably selfish strategies among agents in terms of combat, forag-

ing and niche-finding (Suarez et al. 2023). Assuming that designers have a fixed idea of desirable level features, PCGRL is naturally suited to a cooperative framing, though future work could explore whether novel design patterns emerge when synthetic designers with orthogonal objectives are forced to collaborate.

Yu et al. (2021) investigate the effectiveness of PPO-based RL methods in various cooperative multi-agent settings. They find that PPO-based multi-agent algorithms are competitive with off-policy RL methods such as MADDPG and QMIX. Given this, we find it sufficient to apply Multi-agent PPO (MAPPO) to our PCGRL environment (and leave comparison against other algorithms to future work).

JAX (Bradbury et al. 2018) is a library that allows for a broad array of numpy-type operations (Harris et al. 2020) to be just-in-time compiled and run on the GPU, taking advantage of array parallelism. It has been used to accelerate RL environments in both single-agent (Coward, Beukman, and Foerster 2024), and multi-agent settings (Rutherford et al. 2024). PCGRL has been ported to JAX (Earle, Jiang, and Togelius 2024), allowing for a $\approx 17\times$ speedup in training time relative to the numpy version (Khalifa et al. 2020).

The idea of multi-agent or co-creative PCG has been explored in a few different domains, from using hand-written agents for city planning (Lechner et al. 2003), to co-creative design of perfume bottles (Guzdial, Liao, and Riedl 2018), to exploring a variety of generative models, ranging from locally to globally focused, for co-creative PCGML for level designs in Super Mario Bros. (Quanz et al. 2020). This work is the first, to our knowledge, that treats PCG as a multi-agent RL problem.

Methods

We extend previous work by adding multi-agent support to the JAX version of the PCGRL framework, focusing on the *binary* and *dungeon* domains using the *turtle* representation from (Khalifa et al. 2020).

In the *binary* domain, 'turtle' agents can place impassable 'wall' tiles or traversible 'air' tiles, or move to adjacent positions. At the start of each episode, agents are randomly positioned on the map, which is initialized uniformly with a mixture of wall and air tiles. The objective of the generator agents is to construct a maze with maximal diameter (defined as the longest shortest path between any two air tiles) while maintaining only one connected traversible component. Diameter is approximated using two passes of Dijkstra's algorithm. Agents are rewarded based on how much their actions reduce the distance between the values of these metrics in the current state and their target values (viz. the maximum possible diameter, and 1 connected component).

In the *dungeon* domain, additional constraints are introduced relative to the binary maze domain. Each level includes a player, a key, a door, and several enemies. In a dungeon, the implied goal of the player would be to collect the key, reach the door, and avoid enemies. So for a level to be considered playable, it must contain exactly one player, one key, and one door. Three shortest paths are computed: from the player to the key, from the key to the door, and from the player to the nearest enemy. These paths are computed

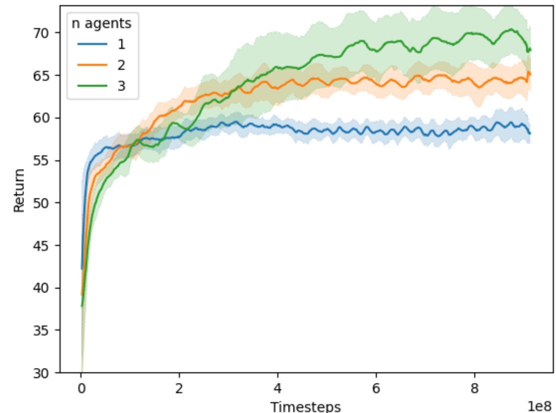

Figure 3: Episode returns by total number of environment steps during training, for varying numbers of agents. Results averaged over 10 training seeds.

only after the correct number of game elements is present. The nearest enemy must be at least three tiles away from the player, and the player's two-part path to unlocking the door should be maximal. The dungeon domain introduces more complex gameplay constraints compared to the binary domain, allowing us to evaluate the agents in scenarios that more closely resemble real-world use cases with diverse requirements and constraints.

To enable more meaningful comparisons across hyperparameter sweeps and experimental settings, we standardize the number of environment steps allowed in an episode using a unit we call a "board scan." A board scan represents the number of steps a single agent would need to traverse the entire board using an optimally space-filling trajectory (e.g., following a *turtle* pattern). For instance, on a 16×16 map, one board scan corresponds to $256 \times 2 = 512$ steps, allocating half the steps for movement and the other half for editing each tile along the path. When this value of maximum allowed board scans is less than 1, in the single agent case, the agent cannot edit all tiles on the map and is thereby forced to incorporate some of the initial noise into its design, which constraint we would expect to discourage mode collapse or memorization of a single optimal level layout.

Reward

The step function effectively applies a batch of tile-edit actions—one per agent—to the level in sequence. (Conflicts resulting from agents occupying the same tile are resolved according to agent ordering.) We implement a shared reward, in which all agents receive a reward corresponding to the degree to which the cumulative edits made by the population of agents bring the level closer to the desired heuristics.

Multi-Agent Environment

Each agent observes a local patch of the map centered at its current position (padded with special "border" tiles when

		mean episode reward							
	map shape	fixed				random			
	map width	8	16	24	32	8	16	24	32
n. agents	max. board scans								
1	1.50	**14.60 ± 0.63**	**42.49 ± 1.20**	**78.85 ± 1.83**	120.65 ± 1.90	5.81 ± 0.44	11.77 ± 0.63	10.25 ± 0.53	16.53 ± 0.39
2	0.75	14.13 ± 0.92	40.97 ± 1.63	72.47 ± 2.75	**114.47 ± 3.38**	**7.47 ± 0.12**	**15.29 ± 0.36**	**22.25 ± 0.75**	21.55 ± 1.34
3	0.50	11.89 ± 0.54	36.24 ± 3.64	67.64 ± 4.35	108.19 ± 2.85	6.53 ± 0.26	11.36 ± 0.19	20.95 ± 1.34	**34.80 ± 0.27**
1	3.0	**16.31 ± 0.43**	44.68 ± 0.42	79.49 ± 2.42	113.19 ± 9.54	7.39 ± 0.11	13.47 ± 0.53	10.41 ± 0.50	16.00 ± 0.28
2	1.5	14.71 ± 0.07	**46.23 ± 1.40**	**87.14 ± 2.18**	**130.92 ± 3.49**	7.32 ± 0.33	14.17 ± 0.22	18.81 ± 0.78	30.29 ± 1.30
3	1.0	15.56 ± 0.63	44.49 ± 1.99	83.94 ± 5.12	124.95 ± 10.47	**8.66 ± 0.47**	**14.82 ± 0.41**	**25.55 ± 1.58**	**41.52 ± 2.81**

Table 2: When controlling the number of environment steps (relative to the time it would take a single agent to scan through each tile on the map), so that the same number of actions are taken in situations with variable number of agents, settings with more agents generally attain comparable reward to those with fewer or a single agent, and do better in out-of-distribution settings where map shape is randomized at evaluation time. With more agents, reward computation—the bottleneck in the environment—occurs less frequently. Average of 5 training seeds.

			mean episode reward							
		map shape	fixed				random			
		map width	8	16	24	32	8	16	24	32
n. agents	max. board scans	reward freq.								
1	3.0	3	**16.52 ± 0.21**	**46.33 ± 1.17**	82.77 ± 0.90	**125.41 ± 4.51**	7.22 ± 0.25	13.65 ± 0.25	10.35 ± 0.50	15.69 ± 1.52
3	1.0	1	15.56 ± 0.63	44.49 ± 1.99	**83.94 ± 5.12**	124.95 ± 10.47	**8.66 ± 0.47**	**14.82 ± 0.41**	**25.55 ± 1.58**	**41.52 ± 2.81**
1	3.0	2	**16.36 ± 0.60**	**49.19 ± 0.99**	80.81 ± 4.15	114.46 ± 17.52	6.93 ± 0.15	13.04 ± 0.35	10.37 ± 0.39	14.85 ± 0.10
2	1.5	1	14.71 ± 0.07	46.23 ± 1.40	**87.14 ± 2.18**	**130.92 ± 3.49**	**7.32 ± 0.33**	**14.17 ± 0.22**	**18.81 ± 0.78**	**30.29 ± 1.30**

Table 3: When adjusting reward computation frequency and number of board scans in order to hold constant the number of total agent actions and reward computations per episode between single and multi-agent scenarios, multi-agent runs perform slightly worse in-distribution, but generalize better relative to their single agent counterparts. Average of 3 training seeds.

this patch extends beyond the edge of the map). To enable coordination between agents, per-agent binary masks are concatenated channel-wise to the map, each with a 1 on whatever cell the relevant agent is located.

We also introduce a new hyperparameter—*reward frequency*—which allows the user to set the interval, in terms of number of environment steps, between each reward computation. This feature may be generally useful in that it allows more infrequent reward computation, where this reward computation is the bottleneck in the time complexity of the environment, being as it involves path-length computations with complexity $O(N^2)$, where N is the maximum width of a given map. In the present work, it allows us to investigate whether any changes in performance or generalization resulting from additional agents might be attributable merely to increased effective reward sparsity (as all agents act at once and are rewarded together) instead of the multi-agent dynamics themselves.

Experiments

Training

We use Multi-Agent Proximal Policy Optimization (MAPPO) (Yu et al. 2022) to train our agents' shared neural network. We use fully connected RNNs for the Actor and Critic networks, and flatten all observations as input to the networks. All networks are implemented using the Flax framework (Heek et al. 2023). We conduct hyper-parameter sweeps and run the experiments on an HPC cluster, with each experiment executes on a single node equipped with an RTX 8000 GPU. Each experiment runs for 10^9 total environment timesteps.

Observation Space

We experiment with global and local observation windows for agents. In all cases, observation sizes are kept equal between agents. At each time step, each agent either observes the entire map state (a global observation), or a subsection of it (local observation); i.e. an egocentric patch of size $(2N-1) \times (2N-1)$ or $M \times M$ with $M < 2N-1$, respectively. Any observation outside of the map is padded with a "border" tile value only used for padding.

The values of observation windows that we experiment over are $[3, 16, 31]$ where 31 corresponds to a full observation of the map at any point, regardless of the agent's position within it.

As in prior work on PCGRL, we train on maps of 16×16 tiles, allowing for comparisons to prior single-agent approaches. Generally speaking, this size is large enough to allow for complex and varied design strategies while remaining small enough to allow for rapid experimentation over broad sets of hyperparameters.

Evaluation

To assess the in-distribution performance of our agents, we evaluate them 50 times, using different random evaluation seeds, on randomly-initialized 16×16 maps, and record the average of cumulative episode reward. To assess their out-of-distribution performance, following (Earle, Jiang, and To-

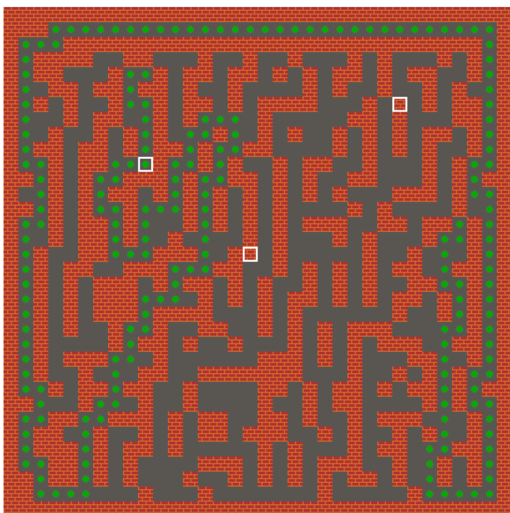

Figure 4: The result of three agents' collaboratively building a binary maze. The agents, trained on 16×16 maps, are evaluated on an out-of-distribution map of size 32×32.

gelius 2024), we evaluate our agents (again with 50 evaluation seeds) using out-of-distribution square and rectangular map sizes. We use square maps of width $8, 16, 24$, and 32. When optionally randomizing map shapes, given one of these widths as the maximum size of the square board, we uniform-randomly sample the width and height of a rectangular map from within these values. This evaluation penalizes agents that overfit by learning one or a handful of globally optimal maps on a given map size, as these "memorized" maps will be either be sub-optimal or impossible to recreate on other map shapes and sizes. In our experiments, each set of hyperparameters is repeated over a number of distinct training seeds. We average results from the evaluation process above applied to models from different training seeds, and report standard deviations in tables (and average and report standard deviations over training metrics from these training seeds in training plots).

Results

Maze Domain

The primary goal of our experiments is to investigate the effect of the number of agents on the performance and generalization ability of agents in a level-generation task.

In Table 1, we examine the effect of varying the number of agents, fixing the number of episode steps to be $2\times$ the number of tiles on the map, and the agent observation windows to be 3×3 egocentric patches. Note that in this experiment, we fix the number of *environment steps*, and in effect the number of reward computations—the most expensive aspect of our environment due to the requisite path-length computations. As a result, settings with more agents are able to make more editing passes on more of the overall map *if* they act collaboratively, and do not merely repeat or undo one another's edits. The results show that scenarios with more agents indeed perform and generalize better.

The performance gains resulting from additional agents can also be seen in the curves reflecting average reward over the course of training, shown in 3.

In Table 2, we adjust the number of environment steps (relative to the total number of tiles in the map, or the maximum number of *board scans* that could be achieved by a single agent given this many steps), so as to hold constant the number of individual tile edits allowed in cases with variable numbers of agents. In effect, scenarios involving more agents are no longer capable of making more editing passes on a larger section of the map relative to single-agent scenarios, as they would have been in Table 1. Thus, any gains resulting from additional agents cannot be attributed to their ability to collectively make more edits to the map over the course of an episode. In this scenario, some of the benefit of multiple agents is reduced relative to Table 7: multi-agent performance and generalization only approaches or is roughly comparable to that of a single agent on small maps with fixed shapes. But significant improvements in generalization still hold in the most complex scenarios, on large maps with random shapes.

In Table 3, we adjust the number of environment steps as well as the reward frequency—the number of steps after which we compute updated metrics and reward—so as to hold constant both the number of individual tile edits, *and* the total number of reward computations, taken over the course of an episode. In this case, any performance or generalization gains resulting from additional agents can be attributed neither to an increase in allowed map edits, nor to the sparsity of reward computations relative to agent edits. Here, similar to Table 2, we note that the quality of generated maps is similar in-distribution and on fixed-shape maps, but again improves significantly on large, randomly-shaped maps given more agents.

Dungeon Domain

In Table 4, we replicate our initial multi-agent result (Table 1) in the dungeon domain. Here, the generator must construct lengthy paths between a player and a key, a key and a door, and a player and a nearest enemy, and arrange for correct numbers of each of these special tile types. In this more complex domain, the advantages of multi-agent collaboration hold.

Supplementary Results

In Table 5, we vary reward frequency alone (in the single-agent case, while fixing the number of steps to be equivalent to one scan of the board, and setting a 3×3 observation window), and find no significant differences in perforamce or generalization.

To validate our choice of local observations in the above experiments, in Table 6 we investigate the effect of observation size in a multi-agent setting, in which 3 agents are allowed to take as many steps in the environment as equate to $1\times$ the number of tiles on the board. We find that smaller, more local observations perform best, with 3×3-tile local observations lead to the best performance and generalization, followed by 16×16 and 32×32 observations.

In Table 7, we conduct a more extensive sweep over number of steps ("board scans"), number of agents, and observation sizes. We can see here that local observations are generally either similarly advantageous or comparable in the 2- and single-agent cases, and that the single-agent case is never dominant in any of the hyperparameter combinations considered.

Discussion

The fact that additional agents results in increased performance in all evaluation settings (including out-of-distribution map shapes and sizes) is strong evidence of the utility of training reinforcement learned level generators in the multi-agent setting. Arguably, these results are somewhat surprising given the additional complexity involved in multi-agent control. In single-agent PCGRL, the next state/observation/reward can be deduced deterministically given the current state and an agent's action. Such a prediction becomes noisy in the multi-agent case, given the stochasticity in the actions of other agents, which is not observed until the following timestep.

Intuitively, when the number of environment steps is held constant, allowing for more agent actions in multi-agent cases, adding agents provides a clear advantage in terms of the amount of map edits and iteration afforded (Tables 1, 4). And because the environment is GPU-accelerated, with agent actions executed in parallel, this increased coverage of the map comes at negligible cost in terms of wall-clock time. But the fact that multi-agent scenarios outperform single-agent ones even when varying the number of environment steps to account for the discrepancy in total number of allowed actions shows that the performance gain cannot be attributed simply to an increased *number* of possible actions.

Even in the case where a single agent can take as many actions as would be taken by multiple agents working together (Table 2), the latter still have a unique advantage in terms of spatial flexibility. Supposing we allow enough actions for either set of agents to edit each tile in the board once, then a single agent must chart a course over the map using a single space-filling curve, whereas multiple agents may combine several smaller such curves to cover the map, allowing for a larger set of possible edit sequences to be enacted upon the map. It may also be the case that by learning to adapt to each others' potentially noisy actions, agents learn more robust policies.

Generally, multi-agent scenarios increase training and run-time efficiency because they require fewer reward computations per agent action. One might expect that such reward sparsity should, if anything, make the task more difficult, forcing agents to make longer time horizon associations between actions and reward. But to be sure that the improved performance and efficiency gains of multi-agent scenarios are not attributable to this reward sparsity, we allow reward computation frequency to be adjusted (Table 3). Comparing multi-agent vs. single-agent settings in which the total number of actions and number of actions between each reward computation are held constant, we find that the generalization gains of multi-agent settings are maintained.

Separately, we note that in the single-agent case, while reward sparsity does not significantly improve performance, it is also not detrimental, up to reward computation intervals of at least 10 agent actions (Table 5). This indicates that there is room to increase reward sparsity (in both single or multi-agent settings), making it possible to scale PCGRL to domains where reward computation is more expensive due to environment size or complexity.

The fact that local observations handily outperform global ones is not surprising, given the empirical precedent revealed by prior work (Earle, Jiang, and Togelius 2024), which suggests that a single feedforward agent is able to communicate globally relevant information to itself across time and space via stigmergy—i.e. implicit signals left by edits to the map. It is nonetheless important to validate that this local approach still provides an advantage, as it is key to scaling learnable environment generation to larger levels where global observations are infeasible. Because in this work we use a recurrent network, it is even less surprising that local observations work well, given that globally relevant facts can be stored in network memory. Communication between agents, on the other hand, must still rely on the stigmergy of clues left in the environment.

Limitations & Future Work

Real-world application of our generative agents will involve levels that are larger, involving more diverse mechanics and a broader array of assets. Often, such tasks are tackled by teams of human designers, comprising multiple specialists.

By framing level design as a multi-agent task, we lay the groundwork for capturing this kind of specialization in teams of automated generators, making large-scale automatic environment generation feasible. In the present work, we observe that some degree of specialization emerges naturally: often, separate agents will stick to different regions of the map, making changes to the level that, when taken on their own, would be locally sub-optimal, but which, when combined with the work of other agents, can push the level toward some global optimum.

Future work could push specialization further. One simple extension could apply the "pinpoint"-tile mechanic of (Earle, Jiang, and Togelius 2024) to any edits made by a chosen *priority* agent, which would then be subsequently unchangeable by other agents. This would enforce specialization and potentially lead to the more rapid emergence of specialized behaviors among agents by explicitly preventing them from overwriting one another's decisions. This may also have the beneficial side-effect of rendering the non-priority agent more adaptable to the actions of a collaborator, working to increase level quality given certain low-level constraints, making for better co-creative assistants that would be more performant when working alongside human designers.

Another common division of labor in human design teams is a hierarchical distribution of design tasks, in which some designers operate at a higher level of abstraction—for example delineating regions, terrain types, or building locations—while other designers focus on lower-level decisions—e.g.

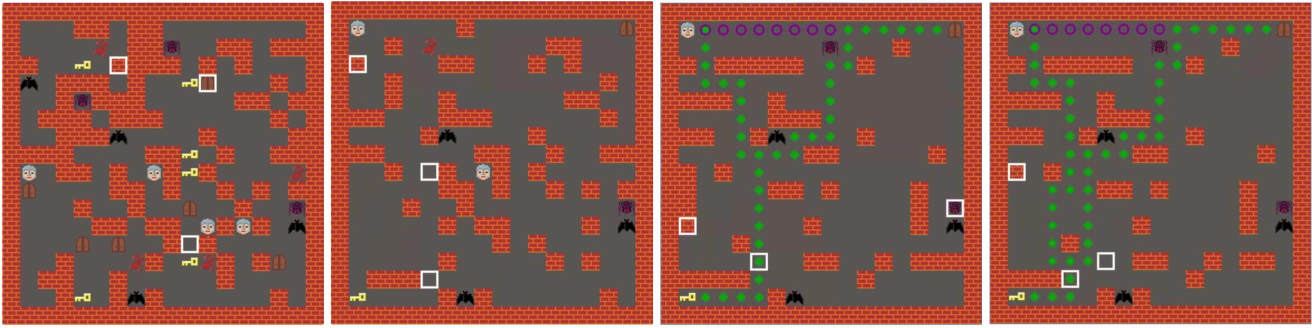

Figure 5: Agents generate a level in the *dungeon* domain.

	mean ep reward							
map width	8		16		24		32	
map shape	fixed	random	fixed	random	fixed	random	fixed	random
n. agents								
1	11.96 ± 0.82	5.53 ± 0.50	66.43 ± 3.35	14.81 ± 0.95	129.77 ± 8.52	34.47 ± 2.30	213.78 ± 17.55	49.33 ± 4.00
2	15.70 ± 0.59	7.68 ± 0.38	92.33 ± 3.47	21.86 ± 1.20	192.70 ± 10.37	51.06 ± 1.14	310.11 ± 29.95	78.06 ± 3.89
3	**16.99 ± 0.59**	**8.52 ± 0.36**	**99.91 ± 3.47**	**24.13 ± 0.95**	**214.17 ± 14.68**	**54.17 ± 3.04**	**364.47 ± 26.86**	**101.63 ± 6.09**

Table 4: Performance of multiple agents when designing levels in the "dungeon" domain. Here as elsewhere, an increase in the number of agents improves performance in and out of distribution while reducing computational cost by reducing the number of reward calculations. Average of 3 training seeds.

placing individual props, laying out paths or defining specific terrain topologies. Multi-agent level design again allows for the instantiation of such a hierarchical work flow. One simple means of realizing this in the current setting could be to afford different agents different "brush stroke" sizes: e.g. allowing one agent to transform entire 3×3 patches of tiles to one type or another, while another agent is able to edit only individual tiles.

We must note that training an RL agent is a time-consuming affair (with the JAX PCGRL framework still requiring several hours—down from days in the numpy version of the framework—to train capable generators), and requires human designers to carefully engineer heuristics that provide learnable signals for agents. If these heuristics are simple enough to be encoded via more classical approaches such as constraint satisfaction, or ample human-authored data is available for supervised learning, then such approaches may provide a faster route to the automatic generation of satisfactory levels.

Conclusion

We frame the problem of collaborative video game level design as a multi-agent reinforcement learning problem, showing how embodied agents with local observations can self-organize over noisy initial levels to iteratively improve them, resulting in globally coherent and functional content. We open-source our code, which builds on an existing, parallelized implementation of the Procedural Content Generation Reinforcement Learning (PCGRL) framework, allowing for rapid multi-agent training on the GPU. While we limit our experimentation to the relatively bare-bones "binary" and "dungeon" PCGRL domains, our method is applicable to any future grid-based domains thay may be added to the framework. It may also be readily combined with existing PCGRL features (such as controllable high-level metrics or frozen tile placements) without further modifications to the code.

Our experiments demonstrate increases in runtime efficiency, performance, and generalization afforded by framing level design as a multi-agent task. We find that even when holding the total number of agent actions constant, distributing these actions among multiple agents leads to stronger generalization ability, particularly on large maps with variable rectangular shapes. We hypothesize that agents in such settings learn robust, modular strategies which are better suited to generalizing to these out-of-distribution scenarios.

Level design is a novel and complex task, distinct from the far more popular game-playing paradigm, that provide a unique set of challenges to multi-agent learning algorithms. The successful automation of such a task has implications not only to the games industry, but also for other domains involving the iterative generation of globally coherent and functional artifacts. Using multi-agent learning opens the door to automating such content generation tasks in a more efficient and scalable manner.

By producing agents that are naturally predisposed for collaboration, we can more readily produce content generators suited for use alongside human designers. Future work should further explore the specialized and modular qualities inherited by multi-agent content generators, either by quantifying emergent behavior of learned generators, or enforcing specialization by design; and validate the utility of these generators in user studies with domain experts.

	mean episode reward							
map shape	fixed				random			
map width	8	16	24	32	8	16	24	32
reward freq.								
1	13.64 ± 0.08	38.85 ± 1.21	68.38 ± 2.19	107.83 ± 1.24	5.82 ± 0.07	8.65 ± 0.16	17.42 ± 0.35	18.53 ± 0.30
2	**13.97 ± 0.24**	38.80 ± 2.01	67.63 ± 2.94	105.48 ± 2.44	**6.37 ± 0.26**	**8.71 ± 0.92**	**17.50 ± 0.69**	18.53 ± 1.20
3	13.59 ± 0.38	37.15 ± 0.97	66.28 ± 2.67	107.05 ± 4.36	6.23 ± 0.32	8.35 ± 0.30	16.86 ± 0.69	18.93 ± 0.66
5	13.07 ± 0.27	38.33 ± 1.24	66.78 ± 3.45	103.51 ± 2.59	6.05 ± 0.13	8.67 ± 0.28	17.17 ± 0.17	18.73 ± 0.25
10	11.44 ± 0.57	**39.59 ± 0.66**	**69.39 ± 1.14**	**109.82 ± 6.81**	4.86 ± 0.26	8.57 ± 0.76	16.65 ± 0.35	**19.25 ± 0.27**

Table 5: Reward computation—the bottleneck of PCGRL complexity—can be performed at least as infrequently as every 10 steps without hurting performance and generalization. However, it does not lead any clear improvement (unlike an increase in the number of agents, which similarly alleviates the need for frequent reward computation). Results for agents given steps-per-episode equal to one single-agent board scan. Average of 3 training seeds.

	mean episode reward							
map shape	fixed				random			
map width	8	16	24	32	8	16	24	32
obs. size								
3	**46.68 ± 1.88**	**133.48 ± 5.96**	**251.82 ± 15.35**	**374.84 ± 31.41**	**25.98 ± 1.40**	**44.46 ± 1.24**	**76.64 ± 4.75**	**124.56 ± 8.42**
16	30.58 ± 2.98	104.44 ± 1.86	185.06 ± 20.51	290.38 ± 33.68	17.32 ± 1.99	32.02 ± 0.67	63.02 ± 3.25	97.56 ± 5.68
31	-1.30 ± 3.00	117.46 ± 3.49	166.12 ± 3.63	260.42 ± 6.81	2.20 ± 2.27	14.46 ± 2.53	48.24 ± 4.55	85.36 ± 1.58

Table 6: In a multi-agent setting—with 3 agents editing the map collaboratively—restricting agents' observation windows to local, 3×3 patches improves performance and generalization. Average of 3 training seeds.

			mean episode reward							
		map shape	fixed				random			
		map width	8	16	24	32	8	16	24	32
max. board scans	n agents	obs size hid dims								
0.75	1	3	12.52 ± 0.47	41.39 ± 2.38	67.25 ± 2.50	96.63 ± 1.40	4.81 ± 0.14	11.77 ± 0.29	12.29 ± 0.88	14.56 ± 0.76
		16	10.50 ± 0.48	45.24 ± 1.28	78.60 ± 0.28	126.78 ± 2.87	3.78 ± 0.04	12.28 ± 0.25	14.16 ± 0.72	17.61 ± 0.45
		31	9.02 ± 2.34	45.37 ± 1.73	54.50 ± 3.46	82.99 ± 8.53	3.05 ± 1.01	11.56 ± 0.77	12.74 ± 0.04	12.73 ± 0.44
	2	3	14.13 ± 0.92	40.97 ± 1.63	72.47 ± 2.75	114.47 ± 3.38	7.47 ± 0.12	15.29 ± 0.36	22.25 ± 0.75	21.55 ± 1.34
		16	11.03 ± 0.28	40.43 ± 2.01	66.75 ± 2.53	92.14 ± 9.73	5.29 ± 0.85	13.21 ± 1.31	21.45 ± 1.54	23.07 ± 2.62
		31	5.57 ± 3.00	44.13 ± 2.10	59.30 ± 3.93	90.77 ± 7.56	3.16 ± 1.10	10.98 ± 1.52	19.87 ± 1.66	19.34 ± 1.43
	3	3	15.07 ± 0.69	**46.51 ± 1.19**	77.09 ± 1.45	119.18 ± 1.28	7.87 ± 0.50	**16.08 ± 0.57**	**26.33 ± 1.91**	29.39 ± 1.75
		16	8.75 ± 3.12	37.62 ± 2.64	67.83 ± 5.33	101.75 ± 7.19	4.11 ± 1.63	12.85 ± 1.86	22.33 ± 2.62	26.23 ± 1.37
		31	1.78 ± 0.48	41.87 ± 0.60	59.19 ± 0.93	90.17 ± 2.70	1.12 ± 0.06	8.57 ± 0.75	20.88 ± 0.82	21.57 ± 1.40
1.0	1	3	13.67 ± 0.14	37.53 ± 1.42	64.03 ± 2.69	101.80 ± 5.91	5.95 ± 0.35	8.04 ± 0.41	16.77 ± 1.28	17.16 ± 0.37
		16	11.01 ± 0.78	43.18 ± 0.50	76.72 ± 3.71	124.57 ± 4.44	5.08 ± 0.48	8.81 ± 0.06	18.56 ± 1.06	21.14 ± 0.24
		31	8.52 ± 2.84	38.78 ± 2.82	50.57 ± 4.03	80.32 ± 7.34	3.89 ± 1.49	8.52 ± 1.53	14.13 ± 1.18	16.03 ± 1.75
	2	3	13.60 ± 0.09	38.81 ± 0.38	68.43 ± 1.45	110.24 ± 3.24	7.28 ± 0.14	11.81 ± 0.38	21.56 ± 1.07	29.45 ± 1.17
		16	11.19 ± 0.70	37.77 ± 1.82	67.14 ± 0.59	97.45 ± 5.44	5.36 ± 0.59	10.46 ± 0.89	21.65 ± 0.75	27.57 ± 2.64
		31	5.01 ± 2.29	42.19 ± 1.69	57.23 ± 1.44	91.31 ± 3.27	2.44 ± 1.38	7.34 ± 0.70	18.23 ± 1.35	25.42 ± 0.94
	3	3	15.56 ± 0.63	44.49 ± 1.99	83.94 ± 5.12	124.95 ± 10.47	**8.66 ± 0.47**	14.82 ± 0.41	25.55 ± 1.58	**41.52 ± 2.81**
		16	10.19 ± 0.99	34.81 ± 0.62	61.69 ± 6.84	96.79 ± 11.23	5.77 ± 0.66	10.67 ± 0.22	21.01 ± 1.08	32.52 ± 1.89
		31	-0.43 ± 1.00	39.15 ± 1.16	55.37 ± 1.21	86.81 ± 2.27	0.73 ± 0.76	4.82 ± 0.84	16.08 ± 1.52	28.45 ± 0.53
1.5	1	3	14.60 ± 0.63	42.49 ± 1.20	78.85 ± 1.83	120.65 ± 1.90	5.81 ± 0.44	11.77 ± 0.63	10.25 ± 0.53	16.53 ± 0.39
		16	12.31 ± 0.59	42.39 ± 2.04	80.93 ± 1.06	119.73 ± 4.15	4.23 ± 0.11	10.71 ± 0.25	9.13 ± 0.42	14.97 ± 0.77
		31	9.91 ± 1.60	39.66 ± 1.57	50.67 ± 3.78	74.19 ± 5.61	3.39 ± 0.95	10.63 ± 0.28	8.37 ± 0.28	12.15 ± 0.47
	2	3	14.71 ± 0.07	46.23 ± 1.40	**87.14 ± 2.18**	**130.92 ± 3.49**	7.32 ± 0.33	14.17 ± 0.22	18.81 ± 0.78	30.29 ± 1.30
		16	11.90 ± 0.31	43.29 ± 0.81	80.72 ± 1.51	113.67 ± 2.34	5.76 ± 0.40	12.93 ± 0.81	17.91 ± 0.34	28.65 ± 1.18
		31	7.98 ± 4.04	42.45 ± 1.96	65.91 ± 1.29	93.65 ± 2.56	3.43 ± 2.33	9.55 ± 2.46	14.42 ± 1.42	23.75 ± 0.43
	3	3	**16.00 ± 0.60**	45.13 ± 0.39	81.44 ± 5.60	119.15 ± 12.59	7.38 ± 0.19	16.08 ± 1.50	23.91 ± 0.74	36.15 ± 3.95
		16	9.92 ± 2.05	36.69 ± 2.42	60.16 ± 5.45	73.66 ± 10.08	4.92 ± 0.67	12.69 ± 1.18	18.31 ± 1.35	27.69 ± 3.21
		31	5.58 ± 4.27	39.39 ± 5.56	60.33 ± 6.62	86.89 ± 7.94	2.69 ± 2.18	7.43 ± 3.84	14.77 ± 3.07	25.43 ± 3.68

Table 7: Extensive sweep over number of agents, number of episode steps (relative to number of steps that would be required by a single agent to perform a scan of the entire board) and observation size. Average of 3 training seeds.

References

Awiszus, M.; Schubert, F.; and Rosenhahn, B. 2020. TOAD-GAN: Coherent Style Level Generation from a Single Example. arXiv:2008.01531.

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; and Zhang, Q. 2018. JAX: composable transformations of Python+NumPy programs.

Buşoniu, L.; Babuška, R.; and De Schutter, B. 2010. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, 183–221.

Charity, M.; Khalifa, A.; and Togelius, J. 2020. Baba is y'all: Collaborative mixed-initiative level design. In *2020 IEEE Conference on Games (CoG)*, 542–549. IEEE.

Coward, S.; Beukman, M.; and Foerster, J. 2024. JaxUED: A simple and useable UED library in Jax. *arXiv preprint arXiv:2403.13091*.

Delarosa, O.; Dong, H.; Ruan, M.; Khalifa, A.; and Togelius, J. 2021. Mixed-initiative level design with rl brush. In *Artificial Intelligence in Music, Sound, Art and Design: 10th International Conference, EvoMUSART 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 10*, 412–426. Springer.

Dennis, M.; Jaques, N.; Vinitsky, E.; Bayen, A.; Russell, S.; Critch, A.; and Levine, S. 2020. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33: 13049–13061.

Earle, S.; Edwards, M.; Khalifa, A.; Bontrager, P.; and Togelius, J. 2021. Learning Controllable Content Generators. *CoRR*, abs/2105.02993.

Earle, S.; Jiang, Z.; and Togelius, J. 2024. Scaling, Control and Generalization in Reinforcement Learning Level Generators. In *2024 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Earle, S.; Parajuli, S.; and Banburski-Fahey, A. 2025. DreamGarden: A Designer Assistant for Growing Games from a Single Prompt. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 1–19.

Ellis, B.; Cook, J.; Moalla, S.; Samvelyan, M.; Sun, M.; Mahajan, A.; Foerster, J.; and Whiteson, S. 2023. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 37567–37593.

Guzdial, M.; Liao, N.; Chen, J.; Chen, S.-Y.; Shah, S.; Shah, V.; Reno, J.; Smith, G.; and Riedl, M. O. 2019. Friend, collaborator, student, manager: How design of an ai-driven game level editor affects creators. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, 1–13.

Guzdial, M.; Liao, N.; and Riedl, M. O. 2018. Co-Creative Level Design via Machine Learning. *CoRR*, abs/1809.09420.

Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; and Oliphant, T. E. 2020. Array programming with NumPy. *Nature*, 585(7825): 357–362.

Heek, J.; Levskaya, A.; Oliver, A.; Ritter, M.; Rondepierre, B.; Steiner, A.; and van Zee, M. 2023. Flax: A neural network library and ecosystem for JAX.

Hüttenrauch, M.; Šošić, A.; and Neumann, G. 2017. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*.

Jiang, M.; Grefenstette, E.; and Rocktäschel, T. 2021. Prioritized level replay. In *International Conference on Machine Learning*, 4940–4950. PMLR.

Jiang, Z.; Earle, S.; Green, M.; and Togelius, J. 2022. Learning controllable 3D level generators. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 1–9.

Khalifa, A.; Bontrager, P.; Earle, S.; and Togelius, J. 2020. PCGRL: Procedural Content Generation via Reinforcement Learning. *CoRR*, abs/2001.09212.

Lechner, T.; Watson, B.; Wilensky, U.; and Felsen, M. 2003. Procedural City Modeling. *SIGGRAPH*.

Merino, T.; Earle, S.; Sudhakaran, R.; Sudhakaran, S.; and Togelius, J. 2024. Making new connections: LLMs as puzzle generators for The New York Times' connections word game. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 20, 87–96.

Merino, T.; Negri, R.; Rajesh, D.; Charity, M.; and Togelius, J. 2023. The Five-Dollar Model: Generating Game Maps and Sprites from Sentence Embeddings. arXiv:2308.04052.

Parker-Holder, J.; Jiang, M.; Dennis, M.; Samvelyan, M.; Foerster, J.; Grefenstette, E.; and Rocktäschel, T. 2022. Evolving Curricula with Regret-Based Environment Design. In *ICML 2022*.

Quanz, B.; Sun, W.; Deshpande, A.; Shah, D.; and Park, J. E. 2020. Machine learning based co-creative design framework. *CoRR*, abs/2001.08791.

Rutherford, A.; Ellis, B.; Gallici, M.; Cook, J.; Lupu, A.; Ingvarsson, G.; Willi, T.; Khan, A.; Schroeder de Witt, C.; Souly, A.; et al. 2024. JaxMARL: Multi-Agent RL Environments and Algorithms in JAX. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2444–2446.

Sarkar, A.; and Cooper, S. 2021. Generating and Blending Game Levels via Quality-Diversity in the Latent Space of a Variational Autoencoder. arXiv:2102.12463.

Schrum, J.; Gutierrez, J.; Volz, V.; Liu, J.; Lucas, S.; and Risi, S. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 148–156.

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

Snodgrass, S.; and Ontanón, S. 2016. Learning to generate video game maps using markov models. *IEEE transactions on computational intelligence and AI in games*, 9(4): 410–422.

Suarez, J.; Bloomin, D.; Choe, K. W.; Li, H. X.; Sullivan, R.; Kanna, N.; Scott, D.; Shuman, R.; Bradley, H.; Castricato, L.; et al. 2023. Neural MMO 2.0: a massively multi-task addition to massively multi-agent learning. *Advances in Neural Information Processing Systems*, 36: 50094–50104.

Sudhakaran, S.; González-Duque, M.; Freiberger, M.; Glanois, C.; Najarro, E.; and Risi, S. 2023. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems*, 36: 54213–54227.

Summerville, A.; and Mateas, M. 2016. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*, 10(3): 257–270.

Todd, G.; Earle, S.; Nasir, M. U.; Green, M. C.; and Togelius, J. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, FDG 2023. ACM.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3): 172–186.

Torrado, R. R.; Khalifa, A.; Green, M. C.; Justesen, N.; Risi, S.; and Togelius, J. 2019. Bootstrapping Conditional GANs for Video Game Level Generation. arXiv:1910.01603.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. arXiv:2103.01955.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; and Wu, Y. 2021. The surprising effectiveness of PPO in cooperative, multi-agent games (2021). *arXiv preprint arXiv:2103.01955*.