Illuminating Mario Scenes in the Latent Space of a Generative Adversarial Network

Matthew C. Fontaine[1], Ruilin Liu[1], Ahmed Khalifa[2], Jignesh Modi[1], Julian Togelius[2], Amy K. Hoover[3], Stefanos Nikolaidis[1]

[1] University of Southern California
[2] New York University
[3] New Jersey Institute of Technology
mfontain@usc.edu, ruilinli@usc.edu, aak538@nyu.edu, jigneshm@usc.edu, julian@togelius.com, ahoover@njit.edu, nikolaid@usc.edu

Abstract

Generative adversarial networks (GANs) are quickly becoming a ubiquitous approach to procedurally generating video game levels. While GAN generated levels are stylistically similar to human-authored examples, human designers often want to explore the generative design space of GANs to extract interesting levels. However, human designers find latent vectors opaque and would rather explore along dimensions the designer specifies, such as number of enemies or obstacles. We propose using state-of-the-art quality diversity algorithms designed to optimize continuous spaces, i.e. MAP-Elites with a directional variation operator and Covariance Matrix Adaptation MAP-Elites, to efficiently explore the latent space of a GAN to extract levels that vary across a set of specified gameplay measures. In the benchmark domain of Super Mario Bros, we demonstrate how designers may specify gameplay measures to our system and extract high-quality (playable) levels with a diverse range of level mechanics, while still maintaining stylistic similarity to human authored examples. An online user study shows how the different mechanics of the automatically generated levels affect subjective ratings of their perceived difficulty and appearance.

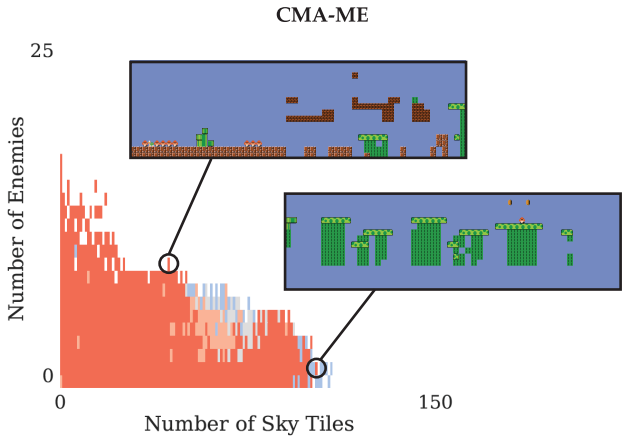


Figure 1: Mario scenes returned by the CMA-ME quality diversity algorithm, as they cover the designer-specified space of two level mechanics: number of enemies and number of tiles above a given height. The color shows the percentage of the level completed by an A* agent, with red indicating full completion.

Introduction

Algorithms that procedurally generate content often need to adhere to a desired style or aesthetics. For example, generative adversarial networks (GANs) (Goodfellow et al. 2014; Karras et al. 2018) generate realistic looking images after training on a large dataset of human specified examples. At the same time, for these algorithms to be useful in practice, they need to enable generation of a *diverse* range of content, across a range of attributes specified by a human designer. For a GAN, this requires either sifting through thousands of randomly generated examples, which is cost-prohibitive, or controlling the GAN output by "steering" it in latent space towards a desired distribution, which is a challenging problem (Jahanian, Chai, and Isola 2020).

When desired attributes can be formulated as an objective, one approach is to explore the latent space using derivative-free optimization algorithms such as CMA-ES (Hansen 2016). Bontrager et al. (2018) named this approach latent variable evolution (LVE). Later, Volz et al. (2018) proposed using GANs to automatically author Mario levels and demonstrated how LVE can extract level scenes with specific attributes from latent space.

The LVE approach is limited to attributes that are easily specifiable as an objective. A human designer may not know *exactly* what kind of content they want, but instead have some intuition on how they would vary content when exploring GAN generated levels. For example, the designer may want to have levels that are of varying difficulty; while it is hard to specify difficulty as an objective, a designer can choose from automatically generated levels of different number of enemies or obstacles.

We call the above problem *latent space illumination* (LSI). Formally, given an objective function and additional functions which measure different aspects of gameplay, we want to extract a collection of game scenes that collectively satisfy all output combinations of the gameplay measures. For each output combination, the representative scene

should maximize the objective function.

Quality diversity (QD) algorithms (Pugh, Soros, and Stanley 2016) are a class of algorithms designed to discover a diverse range of high-quality solutions with several specialized variants designed to explore continuous search spaces.

Our goal in this paper is twofold: First, we wish to find out whether QD algorithms are effective in illuminating the latent space of a GAN, in order to generate high-quality level scenes with a diverse range of desired level characteristics, while still maintaining stylistic similarity to human-authored examples. Second, we want to compare the state-of-the-art QD algorithms in this domain and provide quantitative and qualitative results that illustrate their performance.

A large-scale experiment shows that the QD algorithms MAP-Elites, MAP-Elites (line) and CMA-ME significantly outperform CMA-ES and random search in finding a diverse range of high-quality scenes.[1] Additionally, CMA-ME outperformed the other tested algorithms in terms of diversity and quality of the returned scenes. We show generated scenes, which exhibit an exciting range of mechanics and aesthetics (Fig. 1). A user study shows that the diverse range of level mechanics translates to different subjective ratings of each scenes' difficulty and appearance, highlighting the promise of quality diversity algorithms in generating diverse, high-quality content by searching the latent space of generative adversarial networks.

Background

Procedural Content Generation. Procedural content generation (PCG) refers to creating game content algorithmically, with limited human input (Shaker, Togelius, and Nelson 2016). Game content can be any asset (e.g., game mechanics, rules, dialog, models, etc) required to realize the game for its players. Pioneering work in PCG dates back to the 1980s to address memory limitations for storing large video game levels on computers. The growing interest in realistic graphics in the 1990's necessitated the development of procedural modelling algorithms (Smelik et al. 2014) to generate complex models such as trees and terrain to ease the burden on graphic artists. Much PCG research in both industry and academia has focused on generating playable levels. In general, the problem of generating content that fulfils certain constraints can be approached by evolutionary solutions (Togelius et al. 2011) or constraint satisfaction methods (Smith and Mateas 2011). An emerging area of research is PCG via machine learning (PCGML) which aims to leverage recent advancements in machine learning (ML) to generate new content by treating existing human authored content as training data (Summerville et al. 2018). Previous work in PCGML has enabled automatic generation of video game levels for the Super Mario Bros. using LSTMs (Summerville and Mateas 2016), Markov Chains (Snodgrass and Ontañón 2014) and probabilistic graphical models (Guzdial and Riedl 2016).

Two recent advancements in PCGML are studies by Volz et al. (2018) and Giacomello, Lanzi, and Loiacono (2018) who independently demonstrated the successful application of generative adversarial networks (GANs) to generate playable video game levels in an unsupervised way from existing video game level corpora. Volz et al. (2018) adapted the concept of latent variable evolution (LVE) (Bontrager et al. 2018) to extract Mario scenes from the latent space of a GAN that targeted specific gameplay features. That work searched the latent space of the GAN utilizing the popular Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier 2001) for latent variable inputs that would make the GAN produce level scenes with desired properties. Scenes with targeted gameplay features were obtained through carefully crafted single-objective functions, named fitness functions, that carefully balanced weighted distance from desired gameplay properties on the generated scenes.

Quality Diversity. While the approach employed by Volz et al. (2018) demonstrated a promising synergy between generative models and evolutionary computation for PCG, other works in PCG displayed the potential of quality diversity (QD) to generate meaningfully diverse video game content (Gravina et al. 2019). Unlike traditional optimization methods, QD algorithms aim to generate high quality solutions that differ across specified attributes. Consider the example of generating Mario levels with specific properties. Instead of incorporating the number of enemies or floor tiles into the fitness function, a QD algorithm can treat these measures as attributes. The QD algorithm still has the objective of finding solvable Mario levels, but must find levels that contain all combinations of attributes (number of enemies, percentage of floor coverage). Mouret and Clune (2015) coined the term *illumination algorithms* for quality diversity (QD) algorithms that create an organized mapping between solutions and their associated attributes, which are called behavioral characteristics (BCs). After the QD algorithm generates an organized palette of scenes, stitching algorithms can combine several scenes together to form a cohesive level (Green et al. 2020).

Developed concurrently with our approach is CPPN2GAN (Schrum, Volz, and Risi 2020), which generates full levels for both Super Mario Bros and Zelda. The paper proposes optimizing the latent space of a GAN with a special type of encoding, a compositional pattern producing network (CPPN, (Stanley 2007)), which captures patterns with regularities. The paper introduces a type of latent space illumination with a vanilla version of the quality diversity algorithm MAP-Elites (Mouret and Clune 2015), described in the next section. It focuses on simultaneously searching several latent vectors at once to generate a full level created by "stiching" together GAN-generated scenes. Instead, our focus is on assessing the performance of QD algorithms in generating a variety of scenes with desired characteristics, and in measuring modern MAP-Elites variants that excel at the exploration of continuous domains. Our work is also related with conditional generative models (Hald et al. 2020; Snodgrass and Ontañón 2014; Ping and Dingli 2020). While it is possible to condition GANs on desired BCs, there is no guarantee that the generated scenes will have the properties specified by the conditioning

[1] The source code of the algorithms is available at https://github.com/icaros-usc/MarioGAN-LSI.

input. Additionally, conditional generative models require retraining for each new set of BCs a human designer wishes to explore, where LSI can search the latent space of the same generative model without retraining.

MAP-Elites. MAP-Elites (Mouret and Clune 2015) is a QD algorithm that searches along a set of explicitly defined attributes called behavior characteristics (BCs). These attributes collectively form a Cartesian space named the *behavior space*, which is tessellated into uniformly spaced grid cells. MAP-Elites maintains the highest performing solution for each cell in behavior space (an *elite*) with the product of the algorithm being a diverse archive of high performing solutions. The archive is initially populated with randomly sampled solutions. The algorithm then generates new solutions by selecting elites from the archive at random and perturbing each elite with small variations. The objective of the algorithm is both to expand the archive, maximizing the number of filled cells, and to maximize the quality of the elite within each cell. How the behavior space is tessellated is the focus of a variety of recent algorithms (Smith, Tokarchuk, and Wiggins 2016; Fontaine et al. 2019).

MAP-Elites (line). A common characteristic of many tasks is that high-performing solutions that exhibit diverse behaviors share significant similarities in their "genotype", that is in their search space parameters. Therefore, Vassiliades and Mouret (2018) propose a variational operator, called "Iso+LineDD" which captures correlations between elites. When generating a new solution, in addition to applying a random variation to an existing elite, the operator adds a second random variation directed towards a second elite, essentially nudging the variation distribution towards other high performing solutions. We denote MAP-Elites with this operator ME (line).

CMA-ES. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a second-order derivative-free optimizer for single-objective optimization of continuous spaces (Hansen 2016). The algorithm belongs to a family of algorithms named evolution strategies (ES), which specialize in optimizing continuous spaces by sampling a population of solutions, called a generation of solutions, and gradually moving the population towards areas of highest fitness. CMA-ES models the sampling distribution of the population as a multivariate normal distribution. The algorithm adjusts its sampling distribution by ranking solutions based on their fitness and estimating a new covariance matrix that maximizes the likelihood of future successful search steps.

CMA-ME. The Covariance Matrix Adaptation MAP-Elites (CMA-ME) (Fontaine et al. 2020) is a recent hybrid algorithm which incorporates CMA-ES into MAP-Elites. The algorithm improves the efficiency in which new archive cells are discovered and the overall quality of elites within the archive. CMA-ME maintains a number of individual CMA-ES-like instances, named *emitters*. We use a specific type of emitter named *improvement* emitter, which was shown to outperform MAP-Elites and ME (line) in the strategic card game Hearthstone (Fontaine et al. 2020). Improvement emitters rank solutions by prioritizing those that fill previously undiscovered cells in the archive. Solutions that belong to existing cells in the map are subsequently ranked based on the improvement in fitness over existing cells. This enables improvement emitters to dynamically adapt their goals based on feedback from how the archive changes.

Mario Scene Evaluation

We used the Mario AI Framework[2] to evaluate each of the generated scenes. We evaluate each scene by treating it as a playable level; actual levels are longer and can be generated by "stiching" together multiple scenes (Green et al. 2020).

Following previous work (Volz et al. 2018; Awiszus, Schubert, and Rosenhahn 2020), we approximate playability of a scene by how far through the scene A* reaches; Specifically, we define as "fitness" of a scene the amount of progress by an AI agent playing the scene (percentage of completion in the horizontal direction). We use the A* agent that won the 2009 Mario competition.[3] We additionally define three different types of behavioral characteristics (BCs), which allow for a diverse set of level mechanics.[4]

Representation-Based. We define a set of BCs that capture stylistic aspects of the Mario scene's representation, based on the distribution of tiles. These BCs do not depend on the agent's playthrough:

1. Sky tiles: These are game objects, e.g., blocks, question blocks, coins, that are above a certain height value. A large number implies that there are many game elements above ground, and the player would need to jump to higher tiles.
2. Number of enemies: A larger number of enemies generally results in higher difficulty and requires the player to perform more jumps to navigate throughout the scene.

Agent-Based. We incorporate the agent-based BCs of previous work (Khalifa et al. 2018), which are computed after one playthrough by the agent. The BCs are binary, representing whether the playthrough satisfied a given condition. This results in an 8-dimensional BC-space. The 8 conditions are: (1) performing a jump, (2) performing a high jump (height of jump is above a certain threshold), (3) performing a long jump (horizontal distance is above a certain threshold), (4) stomping on an enemy, (5) killing an enemy using a koopa shell, (6) having an enemy die because of falling out of the scene, (7) collecting a mushroom, and (8) collecting a coin.

KL-Divergence. A common goal in procedural content generation is to generate scenes with different degrees of stylistic similarity to human-designed examples. We use the tile pattern Kullback–Leibler divergence metric (Lucas and Volz 2019) to measure the structural similarity between two Mario scenes. We picked two stylistically different human-designed scenes from the Mario AI Framework, shown in Fig. 2, and we set the behavior characteristics to be the tile pattern KL-divergence between the ground truth scene and generated scene, resulting in a 2-dimensional BC space.

[2] https://github.com/amidos2006/Mario-AI-Framework

[3] https://www.youtube.com/watch?v=DlkMs4ZHHr8

[4] One could also combine the BCs from the three different types, e.g., have an archive with KL-divergence and number of enemies.

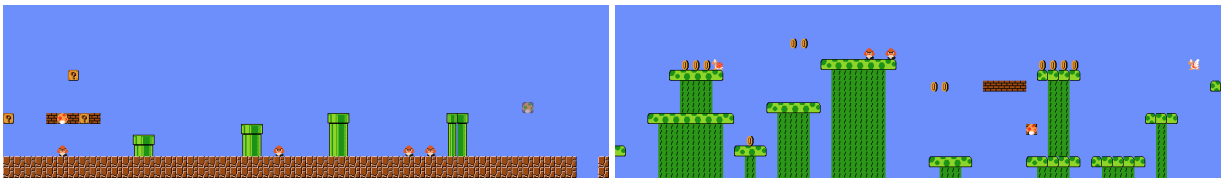


Figure 2: Ground truth scenes 1 (left) and 2 (right) for KL-divergence metric.

Experiments

Our experiments compare the performance of random search, CMA-ES, MAP-Elites, MAP-Elites (line) and CMA-ME on the problem of latent space illumination.

We ran each of the 5 algorithms for 20 trials, 10,000 evaluations each, for each of the three different BC combinations. This resulted in a total of 300 trials. We ran all trials in parallel in a university cluster with multiple nodes running on dual Intel Xeon L5520 processors. Each trial lasted approximately 7 hours.

GAN Model. We use a deep convolutional GAN (DCGAN) as in the study by Volz et al. (2018), trained with the WGAN algorithm (Martin Arjovsky and Bottou 2017). Following their implementation, we encode the training levels by representing each of the 17 different tile types by a distinct integer, which is converted to an one-hot encoded vector, before passed as input to the discriminator. We pad each training level to a 64×64 matrix, and since there are 17 channels, one for each possible tile type, each input scene to the discriminator is $17 \times 64 \times 64$. For the generator, we set the size of the latent vector to be 32, resulting in a 32-dimensional continuous search space. We refer the reader to the study by Volz et al. (2018) for the details of the architecture.

We train the DCGAN with RMSprop for 5000 iterations, a learning rate of $5e^{-5}$ and a batch size of 32. The discriminator iterates 5 times before the generator iterates once. We used for training 15 original levels from the Mario AI competition framework.[5] Fig. 2 shows scenes from two levels of the training data.

To evaluate the different search algorithms, we input the latent vector of size 32 to the generator, and we crop the $17 \times 64 \times 64$ output to a $17 \times 16 \times 56$ playable level for evaluation.

Search Parameters and Tuning. We tuned each algorithm based on how well it covered the representation-based behavior space and we then used the same parameters for all three behavioral characteristics. We set population size $\lambda = 17$ and mutation power $\sigma = 0.5$ for CMA-ES. A single run of CMA-ME deploys 5 improvement emitters with $\lambda = 37$. We set the mutation power for CMA-ME and MAP-Elites $\sigma = 0.2$. For ME (line), we set the isotropic mutation $\sigma_1 = 0.02$ and the mutation for the directional distribution $\sigma_2 = 0.2$. The initial population for MAP-Elites and ME (line) was 100.

In random search, we generate solutions by sampling directly the GAN's latent space from the same distribution that we used to train the generator network: a normal distribution with zero mean and variance equal to 1. We used the same method to generate solutions for the initial population of MAP-Elites and ME (line).

Map Sizes. We performed an initial run of the experiment and we observed the maximum and minimum of values of the behavioral characteristics covered by each algorithm. This provided a rough estimate of the range of each BC.

For the representation-based BCs, we set the range of sky tiles to [0,150] and the number of enemies to [0,25]. The map size was 151×26, where each cell corresponded to an integer value of the BC. The eight agent-based binary BCs form an eight-dimensional map of $2^8 = 256$ cells. Finally, we set the KL-divergence ranges to [0, 4.5] for both groundtruth levels, and the resolution of the map was 60×60.

Metrics. We evaluate all five algorithms, random search, CMA-ES, MAP-Elites, ME (line) and CMA-ME, with respect to the diversity and quality of solutions returned. For comparison purposes, we assign the solutions by CMA-ES and random search to a grid location on what their BC would have been and populate a pseudo-archive.

Percentage of valid cells: This is the percentage of scenes in the archive returned by the algorithm that are completed from start to end by the A* agent, which is equivalent to having a fitness of 1.0. This is an indication of the quality of the solutions found.

Coverage: This is the percentage of cells in the archive produced by an algorithm, computed as the number of cells divided by the total map size. The measure indicates how much of the behavior space is covered.

QD-Score: The QD-Score metric was proposed by Pugh et al. (2015) as the sum of fitness values of all elites in the archive and has become a standard QD performance measure. The measure distills both the diversity and quality of elites in the archive into a single value.

Results

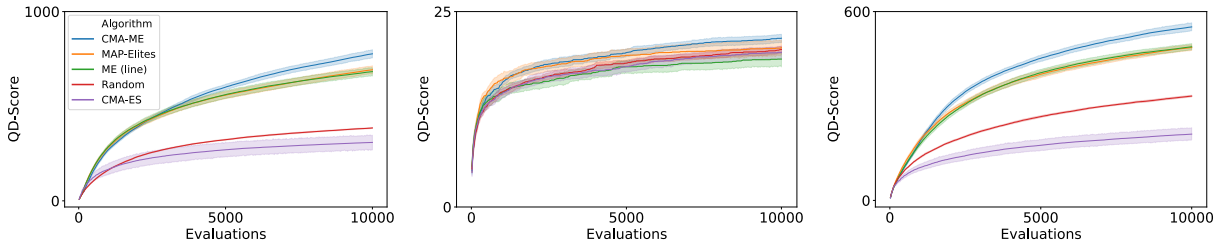
Performance. Table 1 summarizes the performance of each algorithm. Fig. 3 shows improvement in QD-score over evaluations for each algorithm, with 95% confidence intervals.

First, we observe that all QD algorithms, i.e., MAP-Elites, ME (line) and CMA-ME outperform CMA-ES and random search in the representation-based and KL-divergence BCs. This is expected, since CMA-ES optimizes only for one objective, the playability of the scenes, rather than exploring a diverge range of level behaviors. Random search works poorly; the reason is that we sample from the same distribution that we used for training the GAN, thus the gener-

[5] https://github.com/amidos2006/Mario-AI-Framework/tree/master/levels/original

	Representation-Based BCs				Agent-Based BCs				KL-Divergence			
Algorithm	Valid / All	Coverage	Valid / Found	QD-Score	Valid / All	Coverage	Valid / Found	QD-Score	Valid / All	Coverage	Valid / Found	QD-Score
Random	8.35%	11.1%	75.3%	385.1	7.09%	8.9%	79.7%	20.2	5.10%	12.5%	40.8%	331.5
CMA-ES	7.44%	8.0%	93.0%	308.6	7.43%	8.3%	89.6%	19.8	4.11%	7.5%	54.8%	210.6
ME	15.15%	19.4%	78.1%	692.5	7.66%	8.8%	87.0%	20.4	9.98%	15.5%	64.4%	485.6
ME (line)	15.31%	18.9%	81.0%	682.7	7.06%	8.2%	86.1%	18.9	10.18%	15.4%	66.1%	488.0
CMA-ME	16.35%	21.5%	76.1%	776.8	7.90%	9.4%	84.0%	21.6	11.08%	17.4%	63.7%	551.3

Table 1: Results: Average percentage of cells with fitness 1.0 (Valid / All), percentage of cells found (Coverage), percentage of cells found with fitness 1.0 (Valid / Found), and QD-score after 10,000 evaluations.



(a) Representation-based BCs (b) Agent-based BCs (c) KL-divergence

Figure 3: QD-Scores over time for each behavioral characteristic.

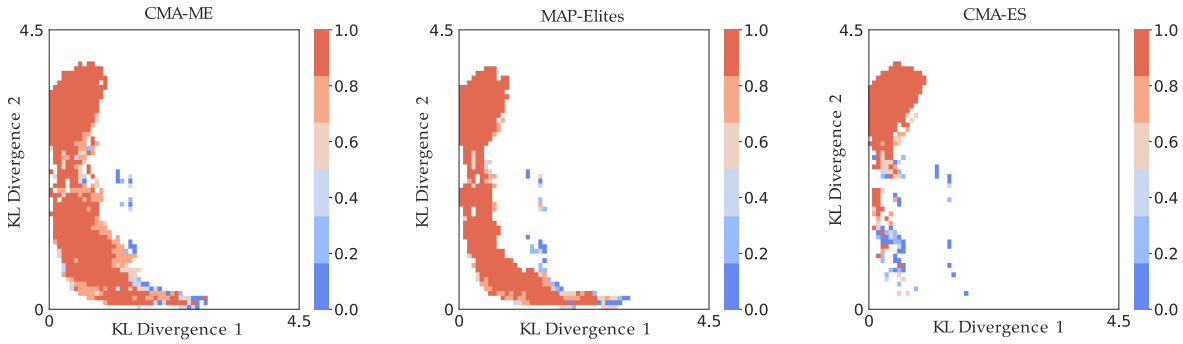


Figure 4: Archive for the KL-divergence behavioral characteristic metric.

ated solutions follow the tile distribution of the training data, which covers only a small portion of the behavior space.

Second, CMA-ME outperforms the other QD algorithms in the representation-based and KL-divergence BCs. This matches previous work (Fontaine et al. 2020), where CMA-ME outperformed these algorithms in the Hearthstone strategic game domain. We attribute this to the fact that CMA-ME benefits by sampling from a dynamically changing Gaussian (as in CMA-ES) rather than a fixed distribution shape. Fig. 4 shows three example archives of elites for CMA-ME, MAP-Elites and CMA-ES, illustrating the ability of CMA-ME to cover larger areas of the map.

We observe that ME (line) performs similarly to MAP-Elites. ME (line) relies on the assumption that different elites in the archive have similar search space parameters. We estimated the similarity of the elite hypervolume as defined in Vassiliades and Mouret (2018), and found low mean values for the representation-based (0.60) and the KL-divergence (0.58) maps, which explains the lack of improvement from the operator in this domain.

On the other hand, in the 8 binary agent-based BCs all algorithms perform similarly to random search. All of the algorithms performed poorly on these BCs, where each algorithm discovers less than 10% of possible mechanic combinations. The main reason lies in the way the A* agent plays the levels; the agent is designed to reach the right edge of the screen as fast as possible, without caring much about its score. This forces the agent to avoid triggering gameplay mechanics. For example, in Fig. 6(right) the agent rushes to the end without collecting the coins in the beginning of the level. The same holds for the training data; the human-authored levels covered only 20 out of the $2^8 = 256$ cells of the map, and there was no training level where the agent collected a mushroom or a coin. This makes the task of finding levels that trigger these BCs even more challenging.

Generated Levels. We demonstrate generated levels by the CMA-ME algorithm that illustrate its ability to generate a diverse range of high-quality solutions.

Figure 5 shows four generated scenes from an archive generated by a single run of CMA-ME using the

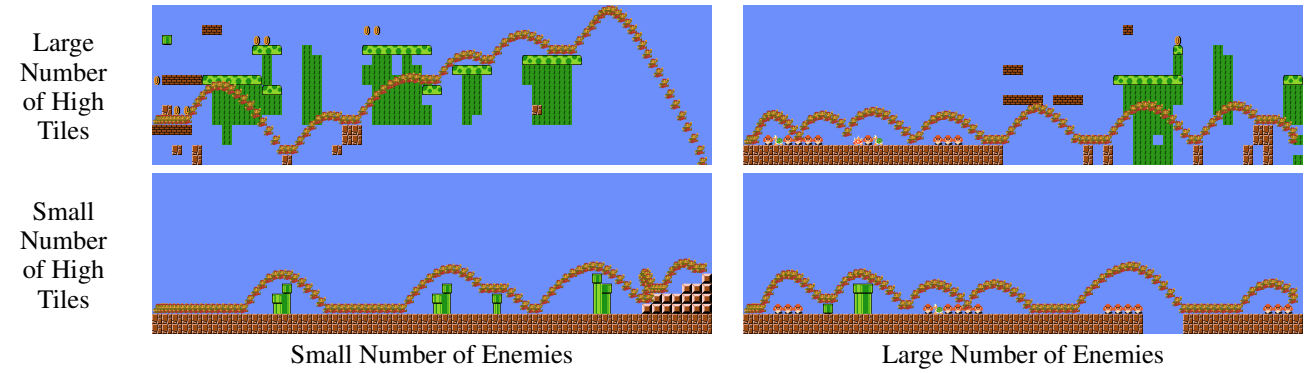


Figure 5: Generated scenes using CMA-ME for small and large values of sky tiles and number of enemies.

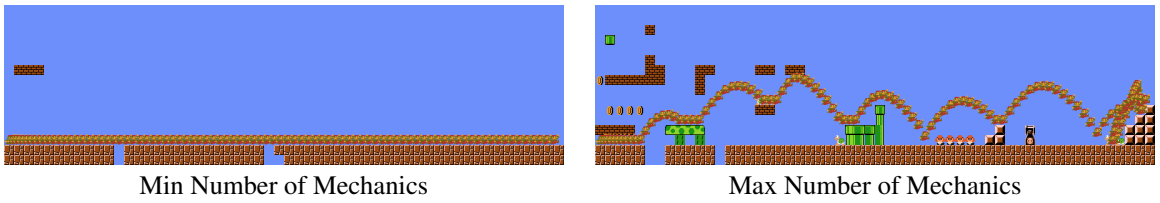


Figure 6: Playable scenes with minimum (left) and maximum (right) sum value (6) of the 8 binary agent-based BCs.

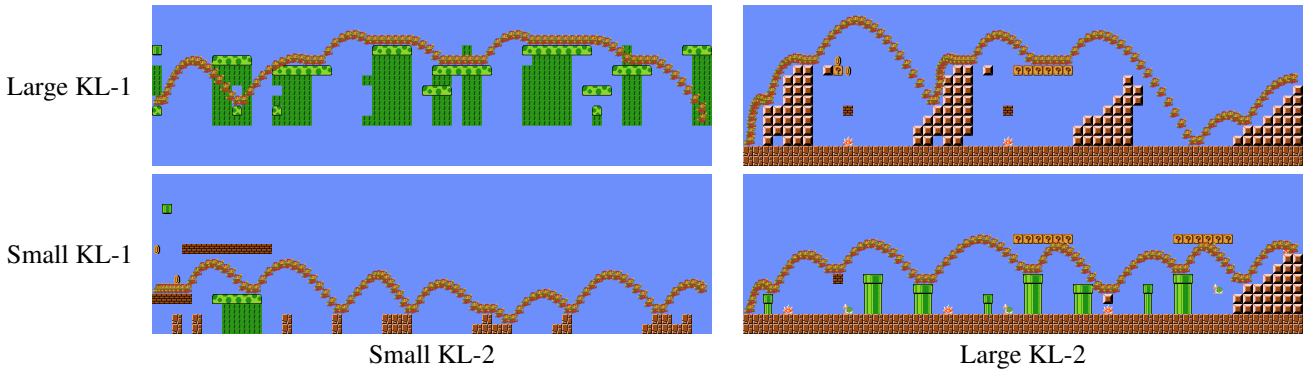


Figure 7: Generated scenes using CMA-ME for small and large values of KL-divergence to each of the two groundtruth scenes.

representation-based BCs. We selected the scenes from the map that had extreme values of the two BCs, the number of sky tiles and number of enemies. The scenes are significantly diverse, with the scene that maximizes each BC being filled with enemies and having multiple tiles above ground. Despite the large number of sky tiles at the level in the top-right, the agent finishes the scene without reaching most of them. This is a limitation of the representation-based BCs, which evaluate a scene based on the distribution of tiles and not on the agent's playthrough.

We address the above limitation with agent-based BCs. Fig. 6 shows two scenes generated by CMA-ME that minimize and maximize the sum of the agent-based BC values. The first scene has 0 value for all BCs and the agent simply runs a straight path towards the exit, while the second scene allows the agent to exhibit a variety of behaviors, including different types of jumps, stomping on an enemy and killing an enemy with a shell.

Finally, Fig. 7 shows four scenes with small and large KL-divergence to each of the two groundtruth scenes in Fig. 2. The scene that is stylistically similar to both groundtruths (bottom-left) combines ground tiles with gaps that force the agent to jump. The top left level maximizes divergence with the first groundtruth scene and minimizes divergence with the second; this results in the scene not having any ground tiles. Interestingly, the scene in the top right maximizes KL-divergence to both groundtruth scene by having tile types and enemies unseen in any of the groundtruth scenes.

User Studies

We have shown how to automatically generate levels that exhibit a diverse range of desired characteristics. Ultimately, the *mechanical* diversity of the generated levels should translate to *perceptual* diversity in how human players per-

ceive the scenes. Our user study is motivated by Sturtevant et al. (2020), which demonstrates how mechanically similar levels can greatly vary in difficulty. Therefore, we conducted two user studies, where we asked users about their perception of scenes generated with the representation-based BCs and the KL-divergence BCs generated with CMA-ME.[6]

Hypotheses.

H1. The perceived difficulty of the generated scenes increases with the number of sky tiles and enemies.

H2. The perceived similarity of the generated scenes relative to the two groundtruth scenes decreases for larger values of KL-divergence.

Scene Difficulty. We expect scenes that that have more sky tiles and larger number of enemies to be perceived as more challenging by human players. We picked 10 scenes uniformly from the representation-based archive, and presented to the users videos of the playthrough of an AI agent for each of the scenes in randomized order.

Dependent Measures. We asked participants to rate how difficult it would be for a human player to complete the scene on a Likert scale from 1 (very easy) to 7 (very hard). At the end of the survey, we also asked them to briefly mention the factors affecting their rating.

Subject Allocation. We recruited human participants through Amazon's Mechanical Turk service, and took several measures to ensure reliability of the results. All participants had approval rate of over 95% and had completed more than 50 tasks. We asked users a control question that tested their attention to the task, and eliminated data associated with a wrong answer, as well as incomplete data. We only considered users that had intermediate or higher experience of playing video games, resulting in 91 samples.

Analysis. We fit a mixed-effects ordinal regression model to the data, with the number of sky tiles and number of enemies as fixed effects and the participant id as random effect. We normalized the numbers of sky tiles and enemies so that their range would be between 0 and 1. We found that both the number of sky tiles ($\beta = 4.35, t(817) = 11.56, p < 0.001$) and number of enemies ($\beta = 0.94, t(817) = 2.48, p = 0.001$) significantly predicted the perceived difficulty of the levels, supporting **H1**. The β values indicate that the number of sky tiles has a stronger effect on the difficulty of the level, compared to the number of enemies. This is because the AI agent attempted to complete the level as fast as possible and it ignored enemies most of the time. Indeed, most participants reported the frequency and length of jumps as the main factor affecting their rating.

Similarity to Groundtruth Scenes. We expect scenes with smaller KL-divergence values to be perceived as more similar to the two groundtruth scenes of Fig. 2. In our second user study, we picked 10 scenes uniformly from the KL-divergence archive and presented them to participants.

Dependent Measures. We asked participants to rate how similar they considered the automatically generated scene to each of the two groundtruth scenes, on a Likert scale from 1 (very different) to 7 (very similar).

[6]Images of the selected scenes and videos of their playthrough by the AI agent are uploaded at: https://icaros-usc.github.io/LSI-Mario-Level-Generation/.

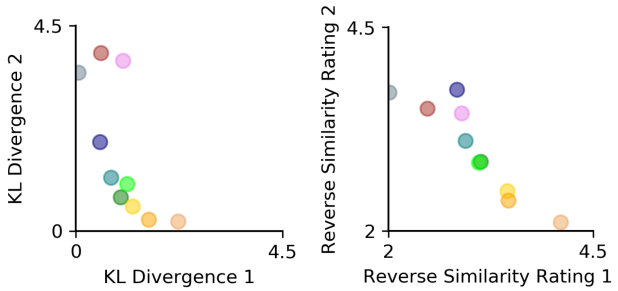


Figure 8: (Left) KL-divergence values of selected level scenes from the KL-divergence archive in Fig. 4-left. Each scene is assigned a unique color in the plot. (Right) Reverse mean ratings of similarity of the same scenes by participants. Level scenes that are identical between the two plots have the same color.

Subject Allocation. We recruited human participants through the Amazon's Mechanical Turk service and followed the same selection process as in the previous study, resulting in 86 samples.

Analysis. We fit two mixed-effects ordinal regression models to the data, one for each of the groundtruth levels, with the KL-divergence to that level as fixed effect and the participant id as random effect. The KL-divergence metric significantly predicted the perceived similarity for both groundtruth scenes ($\beta_1 = -1.24, t(773) = -10.97, p < 0.001$ and $\beta_2 = -0.40, t(773) = -8.84, p < 0.001$), supporting **H2**. Fig. 8 contrasts the KL-divergence metrics of the selected scenes with their (reverse) mean ratings by the participants. While the two plots differ in scale, we see that the metrics capture well the perceived similarity.

Conclusion

We explored the use of QD algorithms to search the latent space of trained generator networks, to create content that has a diverse range of desired characteristics, while retaining the style of human-authored examples. In particular, we described an implementation where the QD algorithms MAP-Elites, MAP-Elites (line) and CMA-ME were used to search the latent space of a DCGAN trained on level scenes from Super Mario Bros. In this problem, CMA-ME was superior to other tested algorithms in terms of coverage and QD-score, indicating that it finds a more diverse and high-quality set of level scenes.

QD algorithms extract a collection of scenes in a single run, rather than just one scene returned by optimization-based methods; their use is thus recommended when a collection of diverse, high-quality content is desired. We are excited about extending this work to search the latent spaces of other generative models, such as variational autoencoders (Doersch 2016) and generative pretraining models (Chen et al. 2020). Finally, we are excited about combining our approach with intelligent trial and error algorithms to create personalized levels (González-Duque et al. 2020).

Acknowledgements

We would like to thank Sebastian Risi for his feedback on a preliminary version of this work.

References

Awiszus, M.; Schubert, F.; and Rosenhahn, B. 2020. TOAD-GAN: coherent style level generation from a single example. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 10–16.

Bontrager, P.; Roy, A.; Togelius, J.; Memon, N.; and Ross, A. 2018. DeepMasterPrints: Generating masterprints for dictionary attacks via latent variable evolution. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 1–9. IEEE.

Chen, M.; Radford, A.; Child, R.; Wu, J.; Jun, H.; Dhariwal, P.; Luan, D.; and Sutskever, I. 2020. Generative Pretraining from Pixels. In *Proceedings of the 37th International Conference on Machine Learning*.

Doersch, C. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* .

Fontaine, M. C.; Lee, S.; Soros, L. B.; de Mesentier Silva, F.; Togelius, J.; and Hoover, A. K. 2019. Mapping Hearthstone Deck Spaces through MAP-Elites with Sliding Boundaries. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, 161–169. New York, NY, USA: ACM. ISBN 978-1-4503-6111-8. doi:10.1145/3321707.3321794. URL http://doi.acm.org/10.1145/3321707.3321794.

Fontaine, M. C.; Togelius, J.; Nikolaidis, S.; and Hoover, A. K. 2020. Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space. *Proceedings of the Genetic and Evolutionary Computation Conference* .

Giacomello, E.; Lanzi, P. L.; and Loiacono, D. 2018. DOOM level generation using generative adversarial networks. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, 316–323. IEEE.

González-Duque, M.; Palm, R. B.; Ha, D.; and Risi, S. 2020. Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error. *arXiv preprint arXiv:2005.07677* .

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, 2672–2680.

Gravina, D.; Khalifa, A.; Liapis, A.; Togelius, J.; and Yannakakis, G. N. 2019. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Green, M. C.; Mugrai, L.; Khalifa, A.; and Togelius, J. 2020. Mario Level Generation From Mechanics Using Scene Stitching. *arXiv preprint arXiv:2002.02992* .

Guzdial, M.; and Riedl, M. O. 2016. Game Level Generation from Gameplay Videos. In *AIIDE*, 44–50.

Hald, A.; Hansen, J. S.; Kristensen, J.; and Burelli, P. 2020. Procedural Content Generation of Puzzle Games using Conditional Generative Adversarial Networks. In *International Conference on the Foundations of Digital Games*, 1–9.

Hansen, N. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* .

Hansen, N.; and Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9(2): 159–195.

Jahanian, A.; Chai, L.; and Isola, P. 2020. On the"steerability" of generative adversarial networks. In *International Conference on Learning Representations (ICLR)*.

Karras, T.; Aila, T.; Laine, S.; and Lehtinen, J. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations (ICLR)*.

Khalifa, A.; Lee, S.; Nealen, A.; and Togelius, J. 2018. Talakat: Bullet Hell Generation Through Constrained Map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, 1047–1054. New York, NY, USA: ACM. ISBN 978-1-4503-5618-3. doi:10.1145/3205455.3205470. URL http://doi.acm.org/10.1145/3205455.3205470.

Lucas, S. M.; and Volz, V. 2019. Tile pattern KL-divergence for analysing and evolving game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 170–178.

Martin Arjovsky, S.; and Bottou, L. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*.

Mouret, J.-B.; and Clune, J. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* .

Ping, K.; and Dingli, L. 2020. Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation. In *Future of Information and Communication Conference*, 400–419. Springer.

Pugh, J. K.; Soros, L. B.; and Stanley, K. O. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* 3: 40. ISSN 2296-9144. doi:10.3389/frobt.2016.00040. URL https://www.frontiersin.org/article/10.3389/frobt.2016.00040.

Pugh, J. K.; Soros, L. B.; Szerlip, P. A.; and Stanley, K. O. 2015. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 967–974.

Schrum, J.; Volz, V.; and Risi, S. 2020. CPPN2GAN: Combining compositional pattern producing networks and gans for large-scale pattern generation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 139–147.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Smelik, R. M.; Tutenel, T.; Bidarra, R.; and Benes, B. 2014. A Survey on Procedural Modelling for Virtual Worlds. *Comput. Graph. Forum* 33(6): 31–50. ISSN 0167-7055. doi: 10.1111/cgf.12276. URL https://doi.org/10.1111/cgf.12276.

Smith, A. M.; and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3): 187–200.

Smith, D.; Tokarchuk, L.; and Wiggins, G. 2016. Rapid phenotypic landscape exploration through hierarchical spatial partitioning. In *International conference on parallel problem solving from nature*, 911–920. Springer.

Snodgrass, S.; and Ontañón, S. 2014. Experiments in map generation using Markov chains. In *FDG*.

Stanley, K. O. 2007. Compositional Pattern Producing Networks: A Novel Abstraction of Development. *Genetic programming and evolvable machines* 8(2): 131–162.

Sturtevant, N.; Decroocq, N.; Tripodi, A.; and Guzdial, M. 2020. The unexpected consequence of incremental design changes. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 130–136.

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.

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.

Vassiliades, V.; and Mouret, J.-B. 2018. Discovering the Elite Hypervolume by Leveraging Interspecies Correlation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 149–156.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 221–228. ACM.