# Procedural Content Generation via Machine Learning (PCGML)

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius

***Abstract*—This survey explores procedural content generation via machine learning (PCGML), defined as the generation of game content using machine learning models trained on existing content. As the importance of PCG for game development increases, researchers explore new avenues for generating high-quality content with or without human involvement; this paper addresses the relatively new paradigm of using machine learning (in contrast with search-based, solver-based, and constructive methods). We focus on what is most often considered functional game content, such as platformer levels, game maps, interactive fiction stories, and cards in collectible card games, as opposed to cosmetic content, such as sprites and sound effects. In addition to using PCG for autonomous generation, cocreativity, mixed-initiative design, and compression, PCGML is suited for repair, critique, and content analysis because of its focus on modeling existing content. We discuss various data sources and representations that affect the generated content. Multiple PCGML methods are covered, including neural networks: long short-term memory networks, autoencoders, and deep convolutional networks; Markov models: $n$-grams and multi-dimensional Markov chains; clustering; and matrix factorization. Finally, we discuss open problems in PCGML, including learning from small data sets, lack of training data, multilayered learning, style-transfer, parameter tuning, and PCG as a game mechanic.**

***Index Terms*—Computational and artificial intelligence, design tools, electronic design methodology, knowledge representation, machine learning, pattern analysis, procedural content generation (PCG).**

## I. Introduction

PROCEDURAL content generation (PCG), the creation of game content through algorithmic means, has become increasingly prominent within both game development and technical games research. It is employed to increase replay value, reduce production cost and effort, to save storage space, or simply as an aesthetic in itself. Academic PCG research addresses these challenges, but also explores how PCG can enable new types of game experiences, including games that can adapt to the player. Researchers also address challenges in computational creativity and ways of increasing our understanding of game design through building formal models [1].

In the games industry, many applications of PCG are what could be called "constructive" methods, using grammars or noise-based algorithms to create content in a pipeline without evaluation. Many other techniques use either search-based methods [2] (for example using evolutionary algorithms) or solver-based methods [3] to generate content in settings that maximize objectives and/or preserve constraints. What these methods have in common is that the algorithms, parameters, constraints, and objectives that create the content are in general hand-crafted by designers or researchers. While it is common to examine existing game content for inspiration, machine learning methods have far less commonly been used to extract data from existing game content in order to create more content.

Concurrently, there has been an explosion in the use of machine learning to train models based on data sets [4]. In particular, the resurgence of neural networks under the name *deep learning* has precipitated a massive increase in the capabilities and application of methods for learning models from big data [5], [6]. Deep learning has been used for a variety of tasks in machine learning, including the generation of content. For example, generative adversarial networks have been applied to generating artifacts, such as images, music, and speech [7]. But many other machine learning methods can also be utilized in a generative role, including $n$-grams, Markov models, autoencoders, and others [8]–[10]. The basic idea is to train a model on instances sampled from some distribution, and then use this model to produce new samples.

This paper is about the nascent idea and practice of generating game content from machine-learned models. We define PCG via machine learning (PCGML) as the generation of game content by models that have been trained on existing game content. The difference to search-based [2] and solver-based [3], [11] PCG is that, while the latter approaches might use machine-learned models (e.g., trained neural networks) for content *evaluation*, the content generation happens through search in *content space*; in PCGML, the content is generated *directly* from the model. By

Manuscript received February 2, 2017; revised August 16, 2017 and January 2, 2018; accepted February 25, 2018. Date of publication June 12, 2018; date of current version September 13, 2018. *(Corresponding author: Julian Togelius.)*

A. Summerville is with the Department of Computational Media, University of California, Santa Cruz, CA 95064 USA (e-mail: asummerv@ucsc.edu).

S. Snodgrass is with the College of Computing and Informatics, Drexel University, Philadelphia, PA 19104 USA (e-mail: sps74@drexel.edu).

M. Guzdial is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: mguzdial3@gatech.edu).

C. Holmgård is with Duck and Cover Games ApS, Copenhagen 1311, Denmark (e-mail: christoffer@holmgard.org).

A. K. Hoover is with the College of Arts, Media and Design, Northeastern University, Boston, MA 02115 USA (e-mail: amy.hoover@gmail.com).

A. Isaksen, A. Nealen, and J. Togelius are with the Department of Computer Science and Engineering, New York University, Brooklyn, NY 11201 USA (e-mail: aisaksen@appabove.com; nealen@nyu.edu; julian@togelius.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TG.2018.2846639

this we mean that the output of a machine-learned model (given inputs that are either drawn from a random distribution or that represent partial or previous game content) is itself interpreted as content, which is not the case in search-based PCG.[1] We can further differentiate PCGML from experience-driven PCG [12] through noting that the learned models are models of game content, not models of player experience, behavior, or preference. Similarly, the learning-based PCG [13] uses machine learning in several roles, but not for modeling content per se.

The content models could be of many different kinds and trained using very different training algorithms, including neural networks, probabilistic models, decision trees, and others. The generation could be partial or complete, autonomous, interactive, or guided. The content could be almost anything in a game, such as levels, maps, items, weapons, quests, characters, rules, etc.

This paper focuses on game content that is directly related to game mechanics. In other words, we focus on *functional* rather than *cosmetic* game content. We define functional content as artifacts that, if they were changed, could alter the in-game effects of a sequence of player actions. The main types of cosmetic game content that we exclude are textures and sound, as those do not directly impact the effects of in-game actions the way levels or rules do in most games, and there is already much research on the generation of such content outside of games [14], [15]. This is not a value judgment, and cosmetic content is extremely important in games; however, it is not the focus of this paper. Togelius *et al.* [2] previously defined a related categorization with the terms necessary and optional. We note that while there exists some overlap between necessary and functional, it is possible to have optional functional content (e.g., optional levels) and necessary cosmetic content (e.g., the images and sound effects of a player character).

It is important to note a key difference between game content generation and procedural generation in many other domains: most game content have strict structural constraints to ensure playability. These constraints differ from the structural constraints of text or music because of the need to play games in order to experience them. Where images, sounds, and in many ways also text can be consumed statically, games are dynamic and must be evaluated through interaction that requires nontrivial effort—in Aarseth's terminology, games are *ergodic media* [16]. A level that structurally prevents players from finishing it is not a good level, even if it's visually attractive; a strategy game map with a strategy-breaking shortcut will not be played even if it has interesting features; a game-breaking card in a collectible card game is merely a curiosity; and so on. Thus, the domain of game content generation poses different challenges from that of other generative domains. Of course, there are many other types of content in other domains that pose different, and in some sense, more difficult challenges, such as lifelike and beautiful images or evocative musical pieces; however, in this paper, we focus on the challenges posed by game content by virtue of its necessity for interaction.

The remainder of this paper is structured as follows. Section II describes the various use cases for PCGML, including various types of generation and uses of the learned models for purposes that are not strictly generative. Section IV-B discusses the key problem of data acquisition and the recurring problem of small data sets. Section III includes a large number of examples of PCGML approaches. As we will see, there is already a large diversity of methodological approaches, but only a limited number of domains have been attempted. In Section IV, we outline a number of important open problems in the research and application of PCGML. Section V concludes this paper.

## II. Use Cases for PCGML

PCGML shares many uses with other forms of PCG: in particular, autonomous generation, cocreation/mixed initiative design, and data compression. However, because it has been trained on existing content, it can extend into new use areas, such as repair and critique/analysis of new content.

### A. Autonomous Generation

The most straightforward application of PCGML is *autonomous PCG*: the generation of complete game artifacts without human input at the time of generation. Autonomous generation is particularly useful when online content generation is needed, such as in rogue-like games.

PCGML is well-suited for autonomous generation because the input to the system can be examples of representative content specified in the content domain. With search-based PCG using a generate-and-test framework, a programmer must specify an algorithm for generating the content and an evaluation function that can validate the fitness of the new artifact [17]. However, designers must use a different domain (code) from the output they wish to generate. With PCGML, a designer can create a set of representative artifacts in the target domain as a model for the generator, and then the algorithm can generate new content in this style. A PCGML avoids the complicated step of experts having to codify their design knowledge and intentions.

### B. Cocreative and Mixed-Initiative Design

A more compelling use case for PCGML is the AI-assisted design, where a human designer and an algorithm work together to create the content. This approach has previously been explored with other methods, such as constraint satisfaction algorithms and evolutionary algorithms [11], [18], [19].

Again, because the designer can train the machine-learning algorithm by providing examples in the target domain, the designer is "speaking the same language" the algorithm requires for input and output. This has the potential to reduce frustration, user error, user training time, and lower the barrier to entry because a programming language is not required to specify generation or acceptance criteria.

PCGML algorithms are provided with example data, and thus are suited to autocomplete game content that is partially

[1]As with any definition, there are corner cases. For example, the functional scaffolding approach to generating levels discussed later in this paper can be described as both search-based PCG and PCGML.

specified by the designer. Within the image domain, we have seen work on image *inpainting*, where a neural network is trained to complete images where parts are missing [20]. Similarly, machine learning methods could be trained to complete partial game content.

### C. Repair

With a library of existing representative content, PCGML algorithms can identify areas that are not playable (e.g., if an unplayable level or impossible rule set has been specified) and offer suggestions for how to fix them. Summerville and Mateas [21] use a special tile that represents where an AI would choose to move the player in their training set, to bias the algorithm towards generating playable content; the system inherently has learned the difference between passable and impassable terrain. Jain *et al.* [22] used a sliding window and an autoencoder to repair illegal level segments—because they did not appear in the training set, the autoencoder replaced them with a nearby window seen during training.

### D. Recognition, Critique, and Analysis

A use case for PCGML that sets it apart from other PCG approaches is its capacity for recognition, analysis, and critique of game content. Given the basic idea of PCGML is to train some kind of model on sets of existing game content, these models could be applied to analyzing other game content, whether created by an algorithm, players, or designers.

Previous work has used supervised training to predict properties of content [23]–[25], but the PCGML enables new approaches operating in an unsupervised manner. Encoding approaches compress the content to an encoded state that can then be analyzed in further processes, such as determining which type of level a piece of content comes from [22] or which levels from one game are closest to the levels from a different game [26].

These learned representations are a byproduct of the generation process, and future work could be used to automatically evaluate game content, as is already done within many applications of search-based PCG, and potentially be used with other generative methods. They also have the potential to identify uniqueness, for example, by noting how frequently a particular pattern appears in the training set, or judging how related a complete content artifact is to an existing set.

### E. Data Compression

One of the original motivations for PCG, particularly in early games, such as *Elite* [27], was data compression. There was not enough space on disk for the game universe. The same is true for some of today's games, such as *No Man's Sky* [28]. The compression of game data into fewer dimensions through machine learning could allow more efficient game content storage. By exploiting the regularities of a large number of content instances, we can store the distinctive features of each more cheaply. Unsupervised learning methods, such as autoencoders, might be particularly well suited to this.

## III. Methods of PCGML

We organize PCGML techniques using the following two dimensions:

1) *Data Representation:* The underlying representation of the data used for training and generation. We consider three representations: sequences, grids, and graphs. We note that it is possible for the same type of content to be represented in many different formats (e.g., platformer levels have been represented as all three representations), and that wildly different content can be represented in the same format (e.g., levels and magic cards as sequences) .
2) *Training Method:* The machine learning technique utilized for training the model. We consider five broad categories of training algorithms: backpropagation, evolution, frequency counting, expectation maximization (EM), and matrix factorization. We note that it is both possible for the same underlying machine learned representation to be trained via different techniques and for two different techniques to utilize the same underlying class of training method (e.g., neural networks can be trained via backpropagation [21], [29], [30] or evolution [31] and EM can be used to train a Bayesian network [32] or *K*-means centroids [33]).

This organization has the benefits of highlighting commonalities across different techniques and game content. Fig. 1 shows a graphical representation of the different approaches that have been attempted. In the following sections, we highlight different PCGML methodologies according to this taxonomy and discuss potential future work to address gaps in the coverage of the prior approaches.

### A. Sequences

Sequences represent a natural format for content that is experienced over time, such as textual content (magic cards) and game levels. We note that the only game levels that have been handled as a sequence have come from the early *Super Mario Bros.* games, where the player can only traverse from left-to-right, meaning that there is a natural ordering of the two-dimensional (2-D) space into a 1-D sequence.

*1) Frequency Counting:* Frequency counting refers to methods wherein the data is split and the frequencies of each type of atomic generative piece (e.g., tile for a tilemap based game) are found, determining the probabilities of generation. These need not simply be the raw frequencies, but are more likely the conditional probability of a piece given some state. Markov chains are a class of techniques that learn conditional probabilities of the next state in a sequence based on the current state. This state can incorporate multiple previous states via the construction of $n$-grams. An $n$-gram model (or $n$-gram for short) is simply an $n$-dimensional array, where the probability of each state is determined by the $n$ states that precede it.

Dahlskog *et al.* trained $n$-gram models on the levels of the original *Super Mario Bros.* game, and used these models to generate new levels [34]. As $n$-gram models are fundamentally 1-D, these levels needed to be converted to strings in order for $n$-grams to be applicable. This was done through dividing

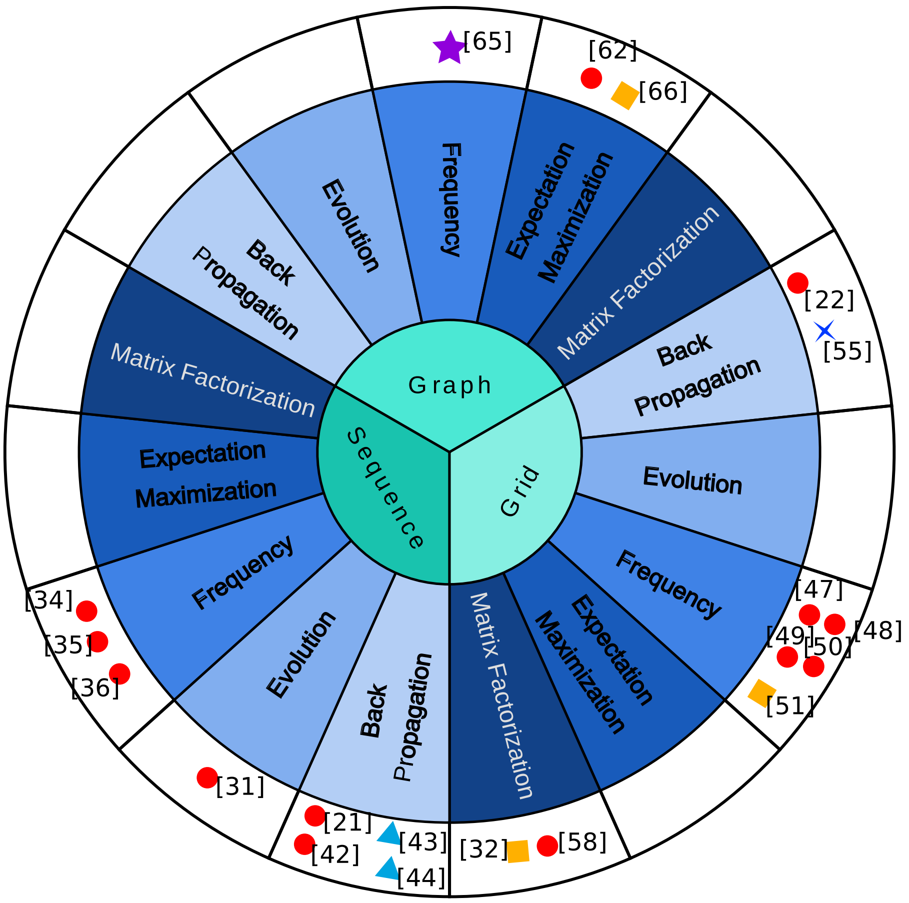


Fig. 1. Our taxonomization of PCGML techniques. We have two categorizations: the underlying data structure (graph, grid, or sequence) and the training method (matrix factorization, EM, frequency counting, evolution, and backpropagation). Marks are colored for the specific type of content that was generated: red circles are platformer levels, orange squares are "dungeons," the dark blue x is real time strategy levels, light blue triangles are collectible game cards, and the purple star is interactive fiction. Citations for each are listed.

the levels into vertical "slices," where most slices recur many times throughout the level [35]. This representational trick is dependent on there being a large amount of redundancy in the level design, something that is true in many games. Models were trained using various levels of $n$, and it was observed that while $n = 0$ creates essentially random structures and $n = 1$ creates barely playable levels, $n = 2$ and $n = 3$ create rather well-shaped levels. See Fig. 2 for examples of this.

Summerville *et al.* [36] extended these models with the use of Monte Carlo tree search (MCTS) to guide generation. Instead of solely relying on the learned conditional probabilities, they used the learned probabilities during roll-outs (generation of whole levels) that were then scored based on an objective function specified by a designer (e.g., allowing them to bias the generation towards more or less difficult levels). The generated levels could still only come from observed configurations, but the utilization of MCTS meant that playability guarantees could be made and allowed for more designer control than just editing of the input corpus.

*2) Evolution:* Evolutionary approaches parameterize solutions as genotypes that are then translated and evaluated in phenotypic or behavior space. This section focuses on evolutionary approaches to generate generators (rather than more

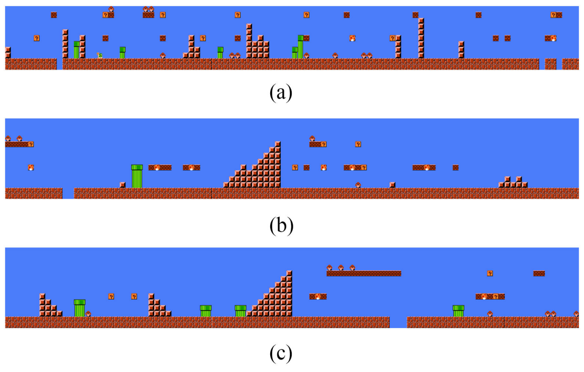


Fig. 2. Mario levels reconstructed by $n$-grams with $n$ set to 1, 2, and 3, respectively. Figures reproduced with permission from [34]. (a) $n = 1$. (b) $n = 2$. (c) $n = 3$.

general content) where the objective function is based on the generator's ability to match a data set.

Hoover *et al.* [31] generate levels for *Super Mario Bros.* by extending a representation called functional scaffolding for musical composition (FSMC) that was originally developed to compose music. The original FSMC evolves musical voices to be played simultaneously with an original human-composed voice [37] via neuroevolution of augmenting topologies [38].

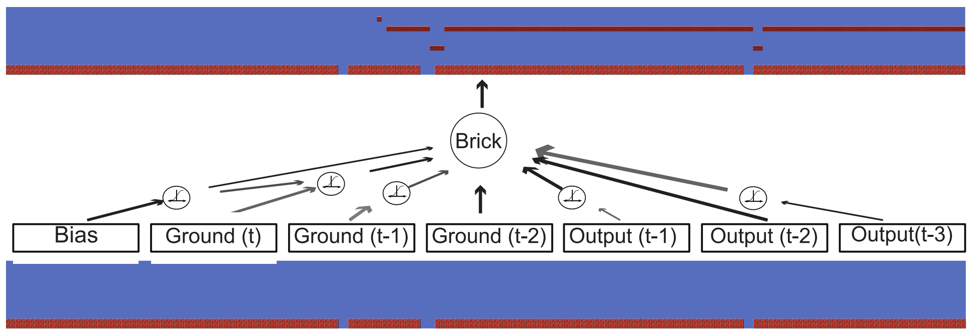


Fig. 3. Visualization of the NeuroEvolution approach, showing the input (bottom), an evolved network architecture, and an example output (top). Figures reproduced with permission from [31].

To extend this musical metaphor and represent *Super Mario Bros.* levels as functions of time, each level is broken down into a sequence of tile-width columns. Additional voices or types of tiles are then evolved with artificial neural networks (ANNs) trained on two-thirds of the existing human-authored levels to predict the value of a tile-type at each column (as shown in Fig. 3). This approach combines a minimal amount of human-authored content with the output from previous iterations. The output of the network is added to the newly created level and fed back as input into the generation of new tile layouts. By acknowledging and iterating on the relationships between design pieces inherent in a human-produced level, this method can generate maps that both adhere to some important aspects of level design while deviating from others in novel ways.

Many evolutionary and evolutionary-like algorithms generate content through machine learning based fitness functions, but because the generation happens through an author-defined search space, they are not PCGML. Another interesting combination of search and machine learning for PCG is the DeLeNoX algorithm, which uses unsupervised learning to continuously reshape the search space and novelty search to search for content to explore the search space [39].

*3) Backpropagation:* Artificial neural networks (ANNs) are universal function approximators and have been used in the field of PCGML as an approximator for a designer. ANNs can be trained via evolutionary techniques as discussed in the previous section, but they are commonly trained via backpropagation. Backpropagation refers to the propagation of errors through the ANN, with each weight in the ANN being changed proportional to the amount of error for which it is responsible. Where the evolutionary approaches are only able to score the entire network, backpropagation tunes parts of the network responsible for errors; however, this comes at the cost that all functions in the ANN must be differentiable (which evolutionary approaches do not require).

Summerville and Mateas [21] used long short-term memory recurrent neural networks (LSTM-RNNs) [40] to generate levels learned from a tile representation of *Super Mario Bros.* and *Super Mario Bros. 2* (JP) [41] levels. LSTMs are a variant of RNNs that represent the current state of the art for sequence-based tasks, able to hold information for hundreds and thousands of time steps, unlike the five to six of standard of RNNs. Summerville and Matteas used a tile representation representing

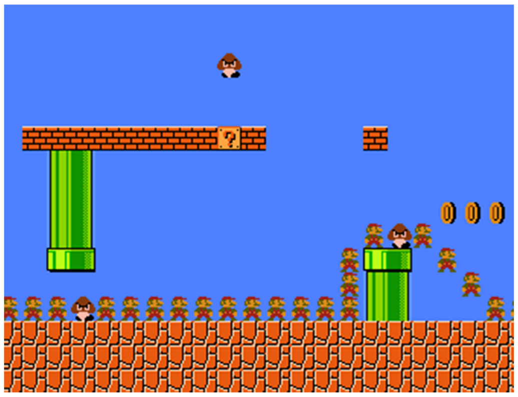


Fig. 4. Example output of the LSTM approach, including generated exemplar player path. Figure reproduced with permission from [21].

levels as a linear string with three different representations used for experimentation. They also included simulated player path information in the input data, forcing the generator to generate exemplar paths in addition to the level geometry. Finally, they included information about how deep into the string the level geometry was, causing the generator to learn both level progression and when a level should end. Fig. 4 shows a segment of generated output for this approach.

Summerville *et al.* [42] extended this work to incorporate actual player paths extracted from four different YouTube playthroughs. The incorporation of a specific player's paths biased the generators to generate content suited to that player (e.g., the player that went out of their way to collect every coin and question-mark block had more coins and question-marks in their generated levels).

A rare example of game content generation that is not a level generator is *Magic: The Gathering* [43] card generation. The first to use a machine learning approach for this is Morgan Milewicz with *@RoboRosewater* [44], a twitter bot that uses LSTMs to generate cards. Trained on the entirety of the corpus of Magic cards, it generates cards, represented as a sequence of text fields (e.g., cost, name, type, etc.). A limitation of this representation and technique is that by treating cards as a sequence, there is no way to condition generation of cards on fields that occur later in the sequence (e.g., Cost occurs near the end of the sequence and as such cannot be used to condition the generation unless all previous fields are specified).

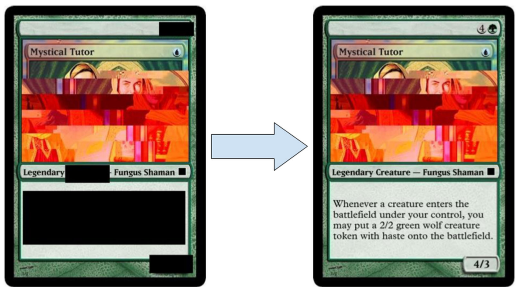

Mystical Tutor
Legendary
Fungus Shaman
Mystical Tutor
Legendary Creature — Fungus Shaman
Whenever a creature enters the battlefield under your control, you may put a 2/2 green wolf creature token with haste onto the battlefield.
4/3

Fig. 5. Partial card specification and the output. Figure reproduced with permission from [30].

Building on the work of *@RoboRosewater* is *Mystical Tutor* by Summerville and Mateas [30]. Using sequence-to-sequence learning [45], wherein an encoding LSTM encodes an input sequence into a fixed length vector that is then decoded by a decoder LSTM, they trained on the corpus of *Magic: The Gathering*. They corrupted the input sequences by replacing lines of text with a `MISSING` token and then tried to reproduce the original cards as the output sequence. The corruption of the input along with the sequence-to-sequence architecture allows the generator to be conditioned on any piece of the card, addressing one of the limitations of *@RoboRosewater*. Fig. 5 shows an example pair of partial specification and the generated output. This work shows multiple different paradigms for PCGML, showing how it can be used on different game content and as a cocreative design assistant, as in Hoover *et al.'s* [31] previously mentioned work.

## B. Grids

Most game levels (particularly of the pre-3-D era) can be thought of as 2-D grids. Sometimes these representations are lossy (e.g., a nontile entity is forcibly aligned to the grid, even if it could actually be at a nontile position), but are generally a natural representation for many different kinds of levels (e.g., platformers, dungeons, real time strategy maps, etc.).

*1) Frequency Counting:* An extension to the previously discussed 1-D Markov chains are multidimensional Markov chains (MdMCs) [46], wherein the state represents a surrounding neighborhood and not just a single linear dimension. Snodgrass and Ontañón [47] present an approach to level generation using MdMCs. An MdMC differs from a standard Markov chain in that it allows for dependencies in multiple directions and from multiple states, whereas a standard Markov chain only allows for dependence on the previous state alone. In their work, Snodgrass and Ontañón represent video game levels as 2-D arrays of tiles representing features in the levels. For example, in *Super Mario Bros.,* they use tiles representing the ground, enemies, *?*-blocks, etc. These tile types are used as the states in the MdMC. That is, the type of the next tile is dependent upon the types of surrounding tiles, given the network structure of the MdMC (i.e., the states that the current state's value depends on).

They train an MdMC by building a probability table according to the frequency of the tiles in training data, given the network structure of the MdMC, the set of training levels, and the set of tile types. A new level is then sampled, one tile at a time, by probabilistically choosing the next tile based upon the types of the previous tiles and the learned probability table.

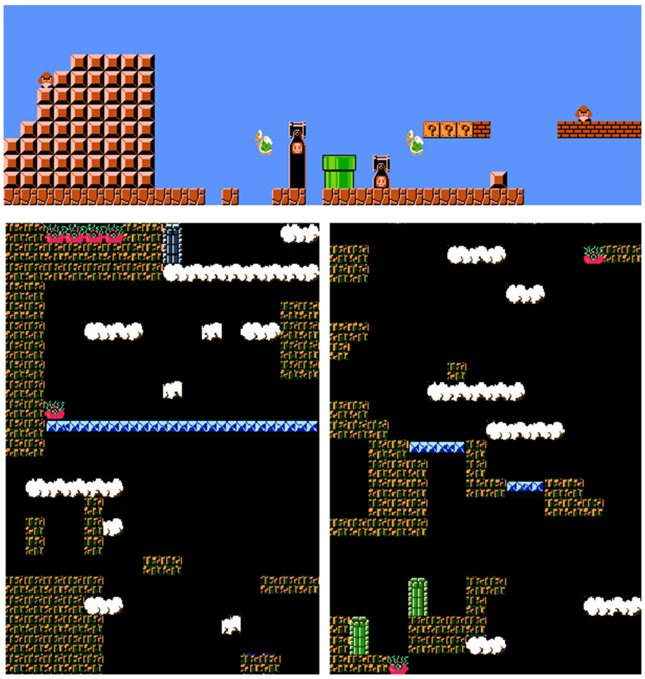


Fig. 6. Sections from a *Super Mario Bros.* level (top) and *Kid Icarus* level section both generated using the constrained MdMC approach (bottom-left), and using an MRF approach (bottom right). Figures reproduced with permission from [48], [49].

In addition to their standard MdMC approach, Snodgrass and Ontañón have explored hierarchical [50] and constrained [48] extensions to MdMCs in order to capture higher level structures and ensure usability of the sampled levels, respectively. They have also developed a Markov random field (MRF) approach [49] that performed better than the standard MdMC model in *Kid Icarus*, a domain where platform placement is pivotal to playability. Fig. 6 shows a section of a *Super Mario Bros.* level (top) and a section of a *Kid Icarus* level (bottom-left) sampled using a constrained MdMC approach, and a section of *Kid Icarus* level (bottom-right) sampled using the MRF approach.

A recent approach by Gumin [51] is loosely inspired by quantum mechanics and uses a "superposition" of tiles to generate images and levels from a representative example tile set. This approach is a variant of MRF, except instead of solely sampling, samples are chosen via "collapsing of the wave function" (i.e., probabilistically choosing a tile and propagating constraints that are choice enforces). This in turn can propagate other changes and either deterministically chooses tiles that no longer have any other possible choices or reduces the possible set of other tiles. The probabilities and configurations are determined by finding each $N \times N$ window in the input, and the number of times that window occurs. This approach was initially explored for bitmap generation, but has since been expanded for use with 3-D tile sets as well as for level generation [52], [53]. The source code and examples of the bitmap project can be found online [51].

*2) Backpropagation:* In [22], Jain *et al.* show how autoencoders [54] may be trained to reproduce levels from the original *Super Mario Bros.* game. The autoencoders are trained on series

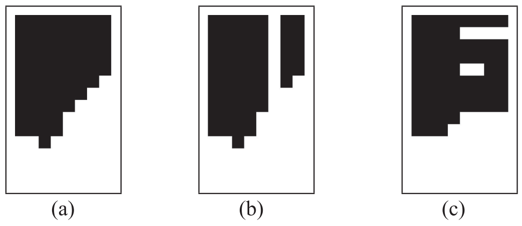


Fig. 7. Original window is overwritten with a wall making the game unplayable. The autoencoder repairs the window to make it playable, although it chooses a different solution to the problem. Figures reproduced with permission from [22]. (a) Original. (b) Unplayable. (c) Repaired.

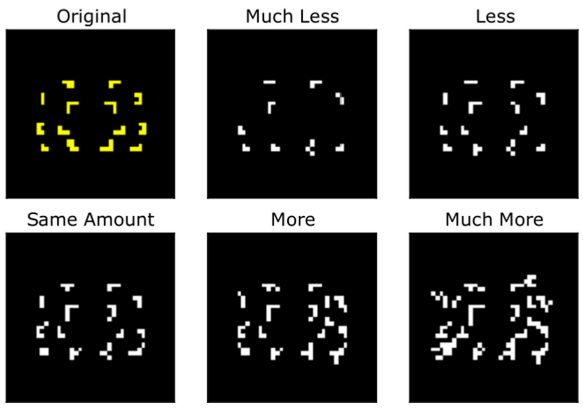


Fig. 8. Varying resource amounts when generating resource locations for *StarCraft II*. Figures reproduced with permission from [55].

of vertical level windows and compress the typical features of Mario levels into a more general representation. They experimented with the width of the level windows and found that four tiles seem to work best for Mario levels. They proceeded to use these networks to discriminate generated levels from original levels, and to generate new levels as transformation from noise. They also demonstrated how a trained autoencoder may be used to repair unplayable levels, changing tile-level features to make levels playable, by inputting a broken/unplayable level to the autoencoder and receiving a repaired one as output (see Fig. 7).

Lee *et al.* [55] use convolutional neural networks to predict resource locations in maps for *StarCraft II* [56]. Resource locations are sparsely represented in *StarCraft II* maps, but are decisive in shaping gameplay. They are usually placed to match the topology of maps. Exploiting this fact, they transform *StarCraft II* maps into heightmaps, downsample them, and train deep neural networks to predict resource locations. The neural networks are shown to perform well in some cases, but struggled in other cases, most likely due to overfitting the small training data set. By adding a number of postprocessing steps to the network, the authors created a tool that allows map designers to vary the amount of resource locations in an automatically decorated map, varying the frequency of mineral placements, shown in Fig. 8.

*3) Matrix Factorization:* Some approaches to level generation find latent level features in high-dimensional data through matrix factorization, which infers features by compressing existing data into a series of smaller matrices. While often generators create levels with a limited expressive range [57], Shaker and Abou-Zleikha [58] create more expressive *Super Mario Bros.* levels by first generating thousands with five known, non-ML-based generators (i.e., notch, parameterized, grammatical evolution, launchpad, and hopper). These levels are then compressed into vectors indicating the content type at each column and transformed into $T$ matrices for each type of content: platforms, hills, gaps, items, and enemies. Through a multiplicative update algorithm [59] for non negative matrix factorization (NNMF), these levels are factored into $T$ approximate “part matrices” that represent level patterns and $T$ coefficient matrices corresponding to weights for each pattern. The part matrices can be examined to see what types of patterns appear globally and uniquely in different generators, and multiplying these part matrices by novel coefficient vectors can be used to generate new levels. This allows the NNMF method to explore far outside the expressive range of the original generators.

While approaches to generating levels typically focus on platformers (e.g., *Super Mario Bros.*), Summerville *et al.* [32] generate levels for *The Legend of Zelda* [60] series. Their approach relies on segmenting data hierarchically by first generating the high-level topological structure of a dungeon (discussed in Section III-C) and then the rooms represented as a grid of tiles via principal component analysis (PCA). The PCA algorithm finds a compressed representation of the original 2-D room arrays by taking the eigenvectors of the original high-dimensional space and retaining only the most informative (inspired by EigenFaces [61]). These compressed representations are represented as weight vectors that can be interpolated between to generate new room content, as seen in Fig. 9.

## C. Graphs

Graphs are the most general data representation considered, with the previous data representations being easily representable as graphs. However, this generality comes at a cost, which is the lack of well-defined structural properties (e.g., in a grid, above and below are implicitly encoded in the structure).

*1) Expectation Maximization:* EM is an iterative algorithm that seeks to find the maximum likelihood estimate or maximum *a posteriori* estimate for the parameters of a model. This is done via a two-step process, first an expectation (E) step wherein the current model’s likelihood is computed given the training data and a maximization (M) step where the parameters are changed to maximize the previously calculated likelihood (which are used in the next E step). An EM is a general training method for any model that is able to provide a likelihood of the model given training data.

$K$-means is a clustering method where an *a priori* defined number of clusters are computed based on the means of the data. Each E step determines which data points belong to which cluster, resulting in new means being calculated for each cluster in the M step. Guzdial and Riedl [62] used a hierarchical clustering approach using $K$-means clustering with automatic $K$ estimation to train their model. They utilized gameplay video of individuals playing through *Super Mario Bros.* to generate new levels. They accomplished this by parsing each *Super Mario Bros.* gameplay video frame-by-frame with OpenCV [63]

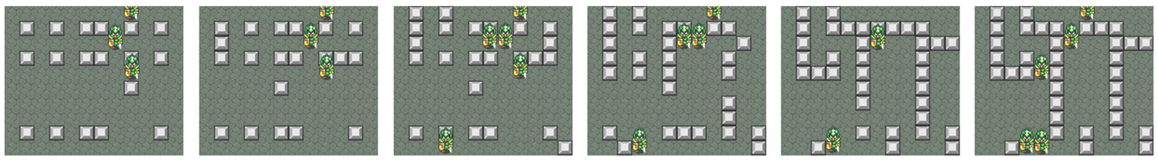


Fig. 9. Example of interpolation between two Zelda rooms (the leftmost and rightmost rooms). Figure reproduced with permission from [32].

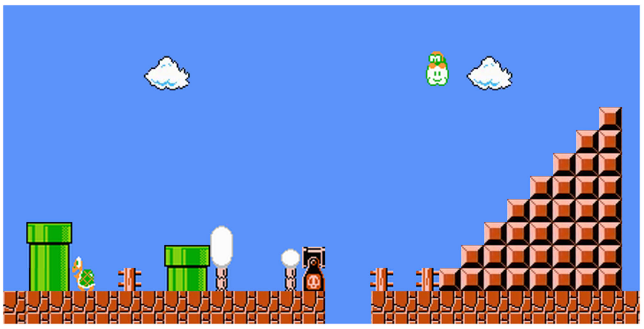
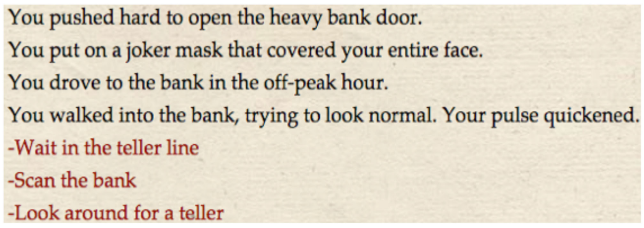


Fig. 10. Level section from the clustering approach. Figure reproduced with permission from [33].



You pushed hard to open the heavy bank door.
You put on a joker mask that covered your entire face.
You drove to the bank in the off-peak hour.
You walked into the bank, trying to look normal. Your pulse quickened.
-Wait in the teller line
-Scan the bank
-Look around for a teller

Fig. 11. Example of Scheherazade-IF gameplay. Figure reproduced with permission from [65].

and a fan-authored spritesheet, a collection of each image that appears in the game. Individual parsed frames could then combine to form chunks of level geometry, which served as the input to the model construction process. In total, Guzdial and Riedl made use of nine gameplay videos for their work with *Super Mario Bros.*, roughly 4 h of gameplay in total.

Guzdial and Riedl's model structure was adapted from [64], a graph structure meant to encode styles of shapes and their probabilistic relationships. The shapes in this case refer to collections of identical sprites tiled over space in different configurations. For further details, please see [62], but it can be understood as a learned shape grammar, identifying individual shapes and probabilistic rules on how to combine them. First, the chunks of level geometry were clustered to derive styles of level chunks, then the shapes within that chunk were clustered again to derive styles of shapes, and finally, the styles of shapes were clustered to determine how they could be combined to form novel chunks of level. After this point, generating a new level requires generating novel level chunks in sequences derived from the gameplay videos. An example screen of generated content using this approach is shown in Fig. 10.

As previously discussed, Summerville *et al.* [66] generated dungeons using two different data representations, with the high-level topological structure defined as a graph with rooms as nodes and linkages (e.g., doors, bombable walls, teleportation portals) as edges. The room-to-room structure of a dungeon is learned along with high-level parameters, such as dungeon size and length of the optimal player path using a Bayes Net [67], a graphical structure of the joint probability distribution. Bayes Nets are trainable via a number of different techniques (e.g., Gibbs sampling, variational message passing), but all are based broadly around maximizing the likelihood of a model given the data. After training the Bayes Net, a designer can "observe" specific parameters of the model, such as how many rooms it should have, how many rooms the player would need to traverse to complete the level, etc. and then sample the rest room by room.

*2) Frequency Counting:* PCGML has focused on graphical games, and particularly the levels of graphical games. However, there exists some work in the field of generating interactive fiction, text-based games like choose your own adventure stories. Guzdial *et al.* adapted Scheherazade [68], a system for automatically learning to generate stories, into Scheherazade-IF [65], which can derive entire interactive fiction games from a data set of stories.

Both Scheherazades rely on exemplar stories crowdsourced from Amazon mechanical turk. The reliance on crowdsourcing rather than finding stories "in the wild" allows Scheherazade to collect linguistically simple stories and stories related to a particular genre or setting (e.g., a bank robbery, a movie date, etc.).

Scheherazade-IF's structures its model as plot graphs [69], directed graphs with events as vertices, and sequentiality information encoded as edges. The system derives events from the individual sentences of the story, learning what events can occur via an adaption of the OPTICS clustering algorithm [70]. The clusters found by OPTICS are found by only allowing points in a cluster if it keeps the density of the cluster the same. Unlike *K*-means, the OPTICS is able to find clusters of arbitrary shape, assuming the shapes are of consistent density. The density is determined by looking in a region around each point and counting the number of other points. The ordering of these primitive events can then be derived from the exemplar stories, which enables the construction of a plot graph for each type of story (e.g., bank robbery). Scheherazade-IF uses these learned plot graphs as the basis for an interactive narrative experience, allowing players to take the role of any of the characters in the plot graph (robber or bank teller in bank robbery) and make choices based on the possible sequences of learned events.

## D. Discussion of Approaches

Broadly, we see that platformer level generation is the most common target of generation and is represented by all types of training and all types of representation. The graph representation is the most general in form [62], with no notion of level shape or size—only the relative positions of entities. The sequence and grid-based approaches all explicitly encode some aspect of the level shape, either by fixing the height at generation time [21], [31], [34], [36], [42] or at run time [26], [47]–[50]—while the matrix factorization approach requires fixing both height and width [58]. Similarly, the convolutional grid approach [29] for RTS map generation requires a fixed map size.

It is hard to compare and contrast the different approaches' performance as there is no agreed upon test for generator "goodness" (i.e., Playability of levels? Capturing of original levels' style? etc.) and the different approaches do not list the time it takes to train or generate a level. However, a few general statements can be made. The LSTM approaches [21], [42] will nearly certainly take more time to train and generate than the MdMC approaches. Both generate one tile at a time, but the LSTM approach inherently requires more computation to generate. Furthermore, the MdMC can be trained in a single pass over the levels, but it is highly unlikely that the training of the LSTM can be stopped after a single training epoch. We note that the matrix factorization approaches have potentially the worst memory usage, as all levels must be held in memory at once. In practice, given the relatively small data sets, this is unlikely a concern, but could pose an issue for situations that call for much larger or many more levels. The backpropagation neural network approaches assume a fixed architecture and as such are the only training size independent approach. In practice, the generator size for the other approaches (MdMC, latent style-graph, evolved neural network, matrix factorized) are likely to be smaller than the back-propagated neural network, but there are no guarantees. A similar concern for the latent style-graph approach is that generation time is dependent on the complexity of the training data (more dense, less regular levels will have more latent nodes, and subsequently more relative position edges) unlike the other approaches that perform the same generation act at each step of generation.

### E. Unexplored Approaches

From Fig. 1, we see that only slightly over half of the possible combinations have been touched on. Notably, evolutionary approaches have only been used to train ANNs for sequences, but given the generality of evolutionary approaches, it seems possible for generators for both graph and grid approaches (given an appropriate objective function). EM approaches are commonly used for training hidden Markov models (HMMs) [71], where the states are hidden from observation and only the output of a hidden state is observed. HMMs seem like a logical extension to the existing Markov chain work, as they allow for higher order structures to be learned instead of just column-to-column transitions. Extending MRFs similarly, conditional random fields [72] act as if the connected grid is based on an unobserved, latent state and have been used in image generation [73].

While it may seem that matrix factorization techniques would be ill-suited to sequences and graphs, those are actually fruitful areas of research. Global Vectors (GloVe) for word representation [74] utilize matrix factorization to embed categorical vocabularies into lower-dimensional real-valued vectors. These vectors are then used for analogical reasoning [75] or for generation [76], but have yet to see use in the generation of game content. Graphs can be represented as adjacency matrices, where the rows and columns are the nodes and each entry in the matrix represents whether nodes are connected (1) or not (0). This enables the field of spectral graph theory, and it seems possible that matrix factorization techniques could learn important properties of graphs represented this way.

Finally, until recently graphs have been an unexplored data structure for backpropagation-based neural networks. Recent work from Kipf and Welling [77] on graph convolutional networks have only been used for classification purposes, but given the lag between pixel convolutional networks being used for classification (1998) [78] and for generation (2015) [79], this seems more a matter of time than a matter of possibility.

## IV. Open Problems and Outlook

In this section, we describe some of the major issues facing PCGML and open areas for active research. Because games as an ML research domain are considerably different from other ML domains, such as vision and audio learning, the problems that arise are commensurately different. In particular, game data sets are typically much smaller than other domains' data sets, they are dynamic systems that rely on interaction, and the availability of high-quality clean data is limited. In the following sections, we discuss the issue of limited data as well as underexplored applications of PCGML for games including style transfer, parameter tuning, and potential PCGML game mechanics.

### A. Ensuring Solvability and Playability

A key aspect of procedural generation for games is the generation of playable artifacts. So far, most of this work has relied on designer specified constraints on the generated works, either via constraints after generation [48] or applying constraints at each step of sampling [36]. While there have been approaches that utilize player paths (either synthetic [21] or actual [42]), these do not ensure playability but instead bias the generators to create playable levels. Future work is required to train a generator to only generate playable levels, but we envision this could be achieved via penalizing the generator at training time, as in generative adversarial networks.

### B. Data Sources and Representations

In this section, we investigate how training data, varying in source, quality, and representation can impact PCGML work. A commonly held view in the machine learning community is that more data is better. Halevy *et al.* [80] make the case that having access to more data, not better algorithms or better data, is the key difference between why some problems are able to be solved in an effective manner with machine learning while others are not (e.g., language translation is a much harder problem than document classification, but more data exists for the translation task). For some types of machine learning, a surfeit of data exists enabling certain techniques, such as images for generative adversarial networks [81] or text for LSTMs-RNN [82]; however, video games do not have this luxury. While many games exist, they typically share no common data structures. Additionally, even if they do, they typically do not share semantics (e.g., sprites in the NES share similarity in how they are displayed but sprite `0x01` is unlikely to have similar semantics between *Super Mario Bros.* [83] and *Metroid* [84], despite the fact that both are platformers). As such, data sources are often limited to within a game series [21], [32] and more commonly within individual games [34], [50], [62]. This in turn means that data scarcity is likely to plague PCGML for the foreseeable future, as there is a small, finite amount of official artifacts (e.g., maps) for most games. However, for many games, there are vibrant fan communities producing hundreds, thousands, and even millions of artifacts, such as maps [85], which could conceivably provide much larger data sources. Within the text of this paper, we discuss approaches to artificially increase the size of data sets through corruption of training data in Magic cards, including multiple player paths for platformer levels, and gathering training data from a secondary source, such as a video.

Recently, Summerville *et al.* created the video game level corpus (VGLC) [86], a collection of video game levels represented in several easily parseable formats. The corpus contains 428 levels from 12 games in 3 different file formats. This marks an initial step in creating a shared data source that can be used for commonality across PCGML research. That said, the repository is limited in scale (12 games and 428 levels is small for standard machine learning) and the representations are lossy (e.g., in Mario, levels only contain a single *Enemy* type mapping for all Goombas, Koopa Troopas, etc.), so additional effort is needed to increase the scope and fidelity of the representations.

Assuming bountiful data existed for video games, bigger questions abound: First, how should the data be represented? Second, what is a training instance? For certain classes of PCGML, such as image generation, obvious data structures exist (e.g., RGB matrices), but there are no common structures for games. Games are a combination of some amount of content (levels, images, 3-D models, text, boards, etc.) and the rules that govern them (What happens when input $x$ is held down in state $y$? What happens if $z$ is clicked on? Can player $\alpha$ move piece $p$? etc.). Furthermore, the content is often dynamic during playtime, so even if the content were to capture with perfect fidelity (e.g., a Goomba starts at tile $X, Y$) how it will affect gameplay requires play (e.g., by the time the player can see the Goomba, it has moved to tile $X', Y'$). Thus, issues of representation serve as a second major challenge, after access to sufficient quantities of quality data.

## C. Learning From Small Data Sets

As previously mentioned, it is a commonly held tenet that more data is better; however, games are likely to always be data-constrained. The English Gigaword corpus [87], which has been used in over 450 papers at the time of writing, is approximately 27 GB. The entire library of games for the Nintendo Entertainment System is approximately 237 MB, over two orders of magnitude smaller. The most common genre, platformers, makes up approximately 14% of the NES library (and is the most common genre for PCG research), which is roughly another order of magnitude smaller. For specific series, it is even worse, with the largest series (*Mega Man* [88]) making up 0.8% of the NES library, for 4 orders of magnitude smaller than a standard language corpus. Standard image corpora are even larger than the Gigaword corpus, such as the ImageNet corpus [89] at 155 GB. All told, work in this area is always going to be constrained by the amount of data, even if more work is put into creating larger, better corpora.

While most machine learning is focused on using large corpora, there is work focused on one-shot learning/generalization [90]. Humans have the ability to generalize very quickly to other objects of the same type after being shown a single object (or very small set, e.g., $n < 5$) and one-shot learning is similarly concerned with learning from a very small data set. Most of the work has been focused on image classification, but games are a natural testbed for such techniques due to the paucity of data.

## D. Learning on Different Levels of Abstraction

As described above, one key challenge that sets PCGML for games apart from PCGML for domains, such as images or sound, is the fact that games are multimodal dynamic systems. This means that content generated will have interactions: generated rules operate on generated levels that in turn change the consequences of those rules and so on. A useful high-level formal model for understanding games as dynamic systems that create experiences is the mechanics, dynamics, aesthetics framework by Hunicke *et al.* [91]. "Mechanics describes the particular components of the game, at the level of data representation and algorithms. Dynamics describes the run-time behavior of the mechanics acting on player inputs and each other's outputs over time. Aesthetics describes the desirable emotional responses evoked in the player, when she interacts with the game system." If the end goal for PCGML for games is to generate content or even complete games with good results at the aesthetic level, the problem is inherently a hierarchical one. It is possible that learning from data will need to happen at all three levels simultaneously in order to successfully generate content for games.

## E. Data Sets and Benchmarks

Procedural content generation, and specifically PCGML is a developing field of research. As such, there are not many publicly available data sets and no widely used standardized evaluation benchmarks.

Publicly available, large data sets are important, as they will reduce the barrier for entry into PCGML, which will allow the community to grow more quickly. As previously mentioned, the VGLC [86] is an important step in creating widely available data sets for PCGML. However, it currently only provides level data. Other data sets exist for object models[2] and gameplay

[2]opengameart.org

mechanics,[3] but they have not been utilized by the PCGML community. By bringing these data sets to the attention of researchers, we hope to promote the exploration and development of approaches for generating those types of content. Additionally, creating, improving, and growing both new and existing data sets are necessary for refining and expanding the field.

Widespread benchmarks allow for the comparison of various techniques in a standardized way. Previously, competitions have been used to compare various level generation approaches [92], [93]. However, many of the competitors in the *Super Mario Bros.* competition included hard coded rules, and the GVG-AI competition provides descriptions of the games, but not the training data necessary for machine learning approaches.

In addition to competitions, there has been some work in developing metric-based benchmarks for comparing techniques. Horn *et al.* [57] proposed a benchmark for *Super Mario Bros.* level generators in a large comparative study of multiple generators, which provided several evaluation metrics that are commonly used now, but only for level generators for platforming games. In recent years, Canossa and Smith [94] presented several general evaluation metrics for procedurally generated levels, which could allow for cross-technique comparisons. Again, however, these only apply to level generation and not other generatable content. Other machine learning domains, such as image processing, have benchmarks in place [78] for testing new techniques and comparing various techniques against each other. For the field of PCGML to grow, we need to be able to compare approaches in a meaningful way.

## *F. Style Transfer*

Style and concept transfer is the idea that information learned from one domain can enhance or supplement knowledge in another domain. Style transfer has most notably been explored for image modeling and generation [95], [96]. For example, in recent work, Gatys *et al.* [97] used neural networks to transfer the style of an artist onto different images. Additionally, Deep Dream [98] trains neural networks on images, and then generates new images that excite particular layers or nodes of the network. This approach could be adapted to learn from game content (e.g., levels, play data, etc.) and generate content that excites layers associated with different elements from the training data. More traditional forms of blending domain knowledge have been applied sparingly to games, such as game creature blending [99] and interactive fiction blending [100]. However, until very recently, no one has applied this machine learning-oriented style transfer to procedural content generation.

Recently Snodgrass and Ontañón [26] explored domain transfer in level generation across platforming games, and Guzdial *et al.* [33] used concept blending to meld different level designs from *Super Mario Bros.* together (e.g., castle, underwater, and overworld levels). These approaches transfer and blend level styles, but do not attempt to address the game mechanics explicitly; both approaches ensure playable levels, but do not attempt transfer or blending between different mechanics. Gow and Corneli [101] proposed a framework for blending two games together into a new game, including mechanics, but no implementation yet exists.

The above approaches are important first steps, but style transfer needs to be explored more fully in PCG. Specifically, approaches to transferring more than just aesthetics are needed, including game mechanic transfer. The above approaches only apply transfer to game levels, but there are many other areas where style transfer can be applied (e.g., character models, stories, and quests).

## *G. Exposing and Exploring the Generative Space*

One avenue that has yet to be deeply explored in PCGML is opening the systems up to allow for designer input. Most systems are trained on a source of data and the editorial control over what data is fed in is the main interaction for a designer, but it is easy to imagine that a designer would want to control things, such as difficulty, complexity, or theming (e.g., the difference between an above ground level and below ground level in *Super Mario Bros.*). The generative approach of Summerville *et al.* [32] allows a designer to set some high-level design parameters (number of rooms in a dungeon, length of optimal player path), but this approach only works if all design factors are known at training time and a Bayes Net is a suitable technique. Similarly, Snodgrass and Ontañón [48] allow the user to define constraints (e.g., number of enemies, distance of the longest gap, etc.), but require hand-crafted constraint checkers for each constraint used.

The nature of generative adversarial networks has allowed for the discovery of latent factors that hold specific semantics (e.g., subtracting the latent space image of blank-faced person from a smiling one is the smile vector) [102]. Interpolating and extrapolating along these latent dimensions allow a user to generate content while freely tuning the parameters they see fit. We envision that this type of approach could lead to similar results in PCG (e.g., subtract Mario 1–1 from Mario 1–2 for the underground dimension, subtract Mario 1–1 from Mario 8–3 for the difficulty dimension). Furthermore, if multiple games are used as training input, it is theoretically possible that one could interpolate between two games (e.g., find the half-way point between Mario and Sonic) or find other more esoteric combinations ($Contra - Mario + Zelda = ???$).

## *H. Using PCGML as a Game Mechanic*

Most current work in PCGML focuses on replicating designed content to provide the player infinite or novel variations on gameplay, informed by past examples. Another possibility is to use PCGML as the main mechanic of a game, presenting the PCGML system as an adversary or toy for the player to engage with. Designs could include enticing the player to, for example, generate content that is significantly similar to or different from the corpus the system was trained on, identify content examples that are outliers or typical examples of the system, or train PCGML systems to generate examples that possess certain qualities or fulfill certain objective functions, teaching the

[3]http://www.squidi.net/three/index.php

player to operate a model by feeding it examples that shape its output in one direction or the other. This would allow a PCGML system to take on several design pattern roles, including AI as role-model, trainee, editable, guided, cocreator, adversary, and spectacle [103].

*Role-model:* A PCGML system replicates content generated by players of various levels of skill or generates content suitable for players of certain skill levels. New players are trained by having them replicate the content or by playing the generated content in a form of generative tutorial.

*Trainee:* The player trains a PCGML system to generate a piece of necessary content (e.g., part of a puzzle or level geometry).

*Editable:* Rather than training the AI to generate the missing puzzle piece via examples, the player changes the internal model's values until acceptable content is generated.

*Guided:* The player corrects the PCG system output in order to fulfill increasingly difficult requirements. The AI, in turn, learns from the player's corrections, following the player's guidance.

*Cocreator:* The player and a PCGML system take turns in creating content, moving toward some external requirement. The PCGML system learns from the player's examples.

*Adversary:* The player produces content that the PCGML system must replicate by generation to survive or vice versa in a "call and response" battle.

*Spectacle:* The PCGML system is trained to replicate patterns that are sensorically impressive or cognitively interesting.

## V. Conclusion

In this survey paper, we give an overview of an emerging machine learning approach to PCG, including describing and contrasting the existing examples of work taking this approach and outlining a number of challenges and opportunities for future research. We intend this paper to play a similar role as the search-based PCG paper [2], which pointed out existing research as well as work that was yet to be done. Much research that was proposed in that paper was subsequently carried out by various authors. There is much work left to do. The vast majority of work has so far concerned 2-D levels, in particular *Super Mario Bros.* levels. Plenty of work remains in applying these methods to other domains, including rulesets, items, characters, and 3-D levels. There is also very rapid progress within machine learning in general in the moment, and in particular within the deep learning field and in methods with generative capabilities, such as generative adversarial networks. There is plenty of interesting and rewarding work to do in exploring how these new capabilities can be adapted to function with the particular constraints and affordances of game content.

## References

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. New York, NY, USA: Springer-Verlag, 2016.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, Sep. 2011.

[3] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 187–200, Sep. 2011.

[4] J. Montgomery, "Machine learning in 2016: What does nips2016 tell us about the field today?" 2016. [Online]. Available: https://blogs.royalsociety.org/in-verba/2016/12/22/machine-learning-in-2016-% what-does-nips2016-tell-us-about-the-field-today/

[5] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[7] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[8] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1881–1888.

[9] S. Fine, Y. Singer, and N. Tishby, "The hierarchical hidden Markov model: Analysis and applications," *Mach. Learn.*, vol. 32, no. 1, pp. 41–62, 1998.

[10] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1462–1471.

[11] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: Reactive planning and constraint solving for mixed-initiative level design," *IEEE Trans. Comput. Intell. AI Games*, vol 3, no. 3, pp. 201–215, Sep. 2011.

[12] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Trans. Affect. Comput.*, vol. 2, no. 3, pp. 147–161, Jul.–Sep. 2011.

[13] J. Roberts and K. Chen, "Learning-based procedural content generation," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 1, pp. 88–101, Mar. 2015.

[14] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, "State of the art in example-based texture synthesis," in *Proc. Eurographics Assoc. State Art Rep.*, 2009, pp. 93–117.

[15] Z. Ren, H. Yeh, and M. C. Lin, "Example-guided physically based modal sound synthesis," *ACM Trans. Graph.*, vol. 32, no. 1, 2013, Art. no. 1.

[16] E. J. Aarseth, *Cybertext: Perspectives on Ergodic Literature*. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1997.

[17] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2010, pp. 141–150.

[18] N. Shaker, M. Shaker, and J. Togelius, "Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels," in *Proc. 9th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2013, pp. 215–216.

[19] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: Computer-aided game level authoring," in *Proc. 8th Int. Conf. Found. Digit. Games*, 2013, pp. 213–220.

[20] C. Guillemot and O. Le Meur, "Image inpainting: Overview and recent advances," *IEEE Signal Process. Mag.*, vol. 31, no. 1, pp. 127–144, Jan. 2014.

[21] A. Summerville and M. Mateas, "Super Mario as a string: Platformer level generation via LSTMs," in *Proc. 1st Int. Joint Conf. DiGRA/FDG*, 2016.

[22] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proc. ICCC Workshop Comput. Creativity Games*, 2016.

[23] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player modeling," in *Dagstuhl Follow-Ups*, vol. 6. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[24] M. Guzdial, N. Sturtevant, and B. Li, "Deep static and dynamic level analysis: A study on infinite Mario," in *Proc. 3rd Exp. AI Games Workshop*, 2016, vol. 3.

[25] A. Summerville, J. R. Mariño, S. Snodgrass, S. Ontañón, and L. H. Lelis, "Understanding Mario: An evaluation of design metrics for platformers," in *Proc. 12th Int. Conf. Found. Digit. Games.*, 2017, Art. no. 8.

[26] S. Snodgrass and S. Ontañón, "An approach to domain transfer in procedural content generation of two-dimensional videogame levels," in *Proc. 12th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2016, pp. 79–85.

[27] D. Braben and I. Bell, *Elite*, 1984.

[28] Hello Games, *No Man's Sky,* 2016.

[29] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Discovering unique game variants," in *Proc. Comput. Creativity Games Workshop ICCC*, 2015.

[30] A. Summerville and M. Mateas, "Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning," in *Proc. 12th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2016, pp. 86–92.

[31] A. K. Hoover, J. Togelius, and G. N. Yannakis, "Composing video game levels with music metaphors through functional scaffolding," *Comp. Creativity Games ICCC*, 2015.

[32] A. Summerville and M. Mateas, "Sampling hyrule: Sampling probabilistic machine learning for level generation," in *Proc. 11th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2015.

[33] M. Guzdial and M. Riedl, "Learning to blend computer game levels," in *Proc. 7th Int. Conf. Comput. Creativity*, 2016, pp. 354–361.

[34] S. Dahlskog, J. Togelius, and M. J. Nelson, "Linear levels through n-grams," in *Proc. Int. Acad. MindTrek Conf.*, 2014, pp. 200–206.

[35] S. Dahlskog and J. Togelius, "Patterns as objectives for level generation," in *Proc. 2nd Workshop Design Patterns Games*, 2013.

[36] A. J. Summerville, S. Philip, and M. Mateas, "MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation," *11th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2015.

[37] A. K. Hoover, P. A. Szerlip, and K. O. Stanley, "Functional scaffolding for composing additional musical voices," *Comput. Music J.*, vol. 38, no. 4, pp. 80–99, 2014.

[38] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, pp. 99–127, 2002.

[39] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis, "Transforming exploratory creativity with delenox," in *Proc. 4th Int. Conf. Comput. Creativity*, 2013, pp. 56–63.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 1997.

[41] S. Miyamoto and T. Tezuka, "Super Mario Bros. 2," 1986.

[42] A. Summerville, M. Guzdial, M. Mateas, and M. Riedl, "Learning player tailored content from observation: Platformer level generation from video traces using LSTMs," in *Proc. 12th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2016.

[43] Wizards of the Coast, *Magic: The Gathering*, 1993.

[44] M. Milewicz, *RoboRosewater*, 2016. [Online]. Available: https://twitter.com/roborosewater?lang=en

[45] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst*, 2014, pp. 3104–3112.

[46] W. Ching, S. Zhang, and M. Ng, "On multi-dimensional Markov chain models," *Pacific J. Optim.*, vol. 3, no. 2, pp. 235–243, 2007.

[47] S. Snodgrass and S. Ontañón, "Experiments in map generation using Markov chains," in *Proc. Found. Digit. Games Conf.*, 2014, vol. 14.

[48] S. Snodgrass and S. Ontañón, "Controllable procedural content generation via constrained multi-dimensional Markov chain sampling," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 780–786.

[49] S. Snodgrass and S. Ontanon, "Learning to generate video game maps using Markov models," *IEEE Trans. Computat. Intell. AI Games*, vol. 9, no. 4, pp. 410–422, Dec. 2017.

[50] S. Snodgrass and S. Ontañón, "A hierarchical MdMC approach to 2D video game map generation," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2015, pp. 205–211.

[51] M. Gumin, "WaveFunctionCollapse," *GitHub Repository*, 2016. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse

[52] Freehold Games, *The Caves of Qud*, 2016.

[53] Arcadia-Clojure, *Proc Skater 2016*, 2016.

[54] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.

[55] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Proc. Artif. Intell. Interactive Digit. Entertainment Conf.*, 2016.

[56] Blizzard Entertainment, *StarCraft II*, 2010.

[57] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the Mario AI framework," in *Proc. Found. Digit. Games Conf.*, 2014.

[58] N. Shaker and M. Abou-Zleikha, "Alone we can do so little, together we can do so much: A combinatorial approach for generating game content," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2014, vol. 14.

[59] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[60] Nintendo, *The Legend of Zelda*, 1986.

[61] M. A. Turk and A. P. Pentland, "Face recognition using eigenfaces," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 1991, pp. 586–591.

[62] M. Guzdial and M. Riedl, "Game level generation from gameplay videos," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2016, pp. 44–50.

[63] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV," *Commun. ACM*, vol. 55, no. 6, pp. 61–69, Jun. 2012.

[64] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun, "A probabilistic model for component-based shape synthesis," *ACM Trans. Graph.*, vol. 31, no. 4, 2014, Art. no. 55.

[65] M. Guzdial, B. Harrison, B. Li, and M. O. Riedl, "Crowdsourcing open interactive narrative," in *Proc. Found. Digit. Games Conf.*, 2015.

[66] A. J. Summerville, M. Behrooz, M. Mateas, and A. Jhala, "The Learning of Zelda: Data-driven learning of level topology," in *Proc. 10th Int. Conf. Found. Digit. Games*, 2015.

[67] S. Wright, "Correlation and causation," *J. Agricultural Res.*, vol. 20, pp. 557–585, 1921.

[68] B. Li, "Learning knowledge to support domain-independent narrative intelligence," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, USA, 2015.

[69] P. Weyhrauch, "Guiding interactive fiction," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 1997.

[70] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, 1999.

[71] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Math. Stat.*, vol. 37, no. 6, pp. 1554–1563, 1966.

[72] J. Lafferty *et al.*, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. 18th Int. Conf. Mach. Learn.*, 2001, vol. 1, pp. 282–289.

[73] X. Liang, B. Zhuo, P. Li, and L. He, "CNN based texture synthesize with semantic segment," arXiv:1605.04731, to be published.

[74] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1532–1543.

[75] B. Eisner, T. Rocktäschel, I. Augenstein, M. Bošnjak, and S. Riedel, "Emoji2vec: Learning emoji representations from their description," in *Proc. 4th Int. Workshop Natural Language Process. Social Media*, 2016, pp. 48–54.

[76] E. Abraham and A. Parrish, "Semantic arithmetic," 2017. [Online]. Available: https://medium.com/dbrs-innovation-labs/semantic-arithmetic-ddae153ca849

[77] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[78] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[79] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1538–1546.

[80] F. Pereira, P. Norvig, and A. Halevy, "The unreasonable effectiveness of data," *IEEE Intell. Syst.*, vol. 24, no. 2, pp. 8–12, Mar./Apr. 2009.

[81] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," arXiv:1611.07004, to be published.

[82] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[83] S. Miyamoto and T. Tezuka, Nintendo, *Super Mario Bros.* 1985.

[84] G. Yokoi and Y. Sakamoto, Nintendo, *Metroid*, 1986.

[85] D. McFerran, "More than a million Super Mario maker levels have been uploaded in a week," 2015. [Online]. Available: http://www.nintendolife.com/news/2015/09/more_than_a_million_super_mari_maker_levels_have_been_uploaded_in_a_week

[86] A. J. Summerville, S. Snodgrass, M. Mateas, and Ontañón, "The VGLC: The video game level corpus," in *Proc. 7th Workshop Procedural Content Gener.*, 2016.

[87] D. Graff, J. Kong, K. Chen, and K. Maeda, "English gigaword," in *Proc. Linguistic Data Consortium*, 2003.

[88] Capcom, *Mega Man*, 1987.

[89] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[90] F.-F. Li, R. Fergus, and P. Perona, "One-shot learning of object categories," *Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, 2006.

[91] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *Proc. AAAI Workshop Challenges Game AI*, vol. 4, 2004, p. 1.

[92] J. Togelius, N. Shaker, S. Karakovskiy, and G. N. Yannakakis, “The Mario AI Championship 2009-2012,” *AI Mag.*, vol. 34, no. 3, pp. 89–92, 2013.

[93] S. L. Ahmed Khalifa, Diego Perez-Liebana, and J. Togelius, “General video game level generation,” in *Proc. Genetic Evol. Comput.*, 2016.

[94] A. Canossa and G. Smith, “Towards a procedural evaluation technique: Metrics for level design,” in *Proc. Conf. Found. Digit. Games*, 2015.

[95] S. Bruckner and M. E. Gröller, “Style transfer functions for illustrative volume rendering,” in *Computer Graphics Forum*, vol. 26, no. 3, 2007, pp. 715–724.

[96] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, “Image analogies,” in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 327–340.

[97] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” arXiv:1508.06576, to be published.

[98] A. Mordvintsev, C. Olah, and M. Tyka, “Inceptionism: Going deeper into neural networks,” *Google Res. Blog*, vol. 20, 2015.

[99] P. Ribeiro, F. C. Pereira, B. Marques, B. Leitao, and A. Cardoso, “A model for creativity in creature generation,” in *Proc. GAME-ON*, 2003, p. 175.

[100] J. Permar and B. Magerko, “A conceptual blending approach to the generation of cognitive scripts for interactive narrative,” in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2013.

[101] J. Gow and J. Corneli, “Towards generating novel games using conceptual blending,” in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2015.

[102] T. White, “Sampling generative networks: Notes on a few effective techniques,” arXiv:1609.04468, to be published.

[103] M. Treanor *et al.*, “AI-based game design patterns,” in *Proc. Conf. Found. Digit. Games*, 2015.

Authors’ photographs and biographies not available at the time of publication.