Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-20)

# Generating Game Levels for Multiple Distinct Games with a Common Latent Space

**Vikram Kumaran, Bradford W. Mott, James C. Lester**
Department of Computer Science, North Carolina State University
{vkumara, bwmott, lester}@ncsu.edu

**Abstract**

Generative adversarial networks (GANs) are showing significant promise for procedural content generation (PCG) of game levels. GAN models generate game levels by mapping a low dimensional latent space to game levels in the game space. An intriguing challenge in GAN-based PCG is enabling GANs to produce game levels for multiple distinct games with similar gameplay characteristics using a common underlying low-dimensional representation. In this paper, we present a method for training a novel GAN-based PCG architecture that generates levels in multiple distinct games, starting from a common gameplay action sequence. We evaluate the solvability of the generated games using an automated playing agent and show how the generated game levels are separate representations of the same gameplay by quantifying the similarity between the solution action sequences for the game levels. By probing the common latent space, we show how our approach provides control over the levels generated in distinct games for the presence of desired gameplay patterns in the generated game levels. Results also demonstrate that the GAN-based PCG approach creates novel game levels in multiple distinct games, as indicated by the distance between the action sequences required to solve the game levels.

## Introduction

Procedural content generation (PCG) holds significant promise for algorithmically creating game content. PCG can be utilized to generate game rules, game levels, and textures for a game's graphical elements. Early work in PCG used search-based and solver-based techniques to generate content, but more recently, machine learning techniques such as deep neural networks have been used to generate game content (Shaker, Togelius, and Nelson 2016). This approach is referred to as Procedural Content Generation via Machine Learning (PCGML) (Summerville et al. 2018; Justesen et al. 2019; Guzdial et al. 2018).



A variety of deep neural network architectures have been used in PCGML to model and generate game content. Game levels in some genres of games such as platformers can be expressed as a sequence of obstacles that players experience as they progress through the game, inspiring the use of Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks, which can encode sequential patterns, to generate new game levels (Summerville et al. 2016; Summerville and Mateas 2016). Generative Adversarial Networks (GAN) are another type of deep neural network that have also gained popularity as a promising content generation technique (Torrado et al. 2019; Volz et al. 2018), driven by the fact that GANs can be trained in an unsupervised fashion given sufficient training examples. However, deep neural networks require a large number of training examples for a model to accurately capture patterns. GANs have sometimes been used to address this requirement by taking a limited set of existing examples to bootstrap a game level training corpus and create more training examples by using their ability to reproduce patterns inherent in training examples when used as a generator (Torrado et al. 2019; Park et al. 2019). However, this approach could lead to overfitting as GANs are learning from a limited set of examples.

A key feature of GANs is the ability to capture game-level patterns in a low dimensional latent representation. Can a single common latent space capture the patterns of game levels from multiple games? In other words, are there underlying shared patterns belonging to multiple distinct games with similar gameplay? We answer this question in the affirmative and explore the implications of the common latent space.

We believe that there is an underlying commonality in game-levels across multiple games despite apparent variability. By capturing the commonality in a novel GAN based model, we create levels in multiple games with similar gameplay from a single common seed. In our work, we use a novel technique to build up the training corpus using an approach similar to the progressive

generation of game levels using action timelines (Shaker et al. 2015). Using this technique that starts from a player's gameplay, we are able to generate sets of game levels in multiple games with similar gameplay and build a sizable training corpus for our GANs. We investigate the implications of a single latent space that captures the combined patterns of multiple distinct games by exploring the relationship between control over the generated levels and the need to build novel solvable game levels. We propose metrics that quantify the novelty of the levels generated by our generator.

## Related Work

Over the years there have been many approaches to capturing a common representation for multiple games. Bentley and Osborn (2019) labeled affordances of sprite patterns in multiple games from the players' point of view, to show commonality between games. Snodgrass et al. (Snodgrass et al. 2016) captured statistical regularities in different platformer game levels. They build the generator using multi-dimensional Markov chains to represent the transition states. Guzdial and Riedl (Guzdial and Riedl 2016; Guzdial and Riedl 2018a; Guzdial and Riedl 2018b; Guzdial, Liao, and Riedl 2018) demonstrate different techniques to combine game concepts and levels to form novel levels, including conceptual expansion, combinatorial creativity and co-creation. Sarkar et al. (Sarkar and Cooper 2018; Sarkar, Yang, and Cooper 2019) have used deep learning models like LSTMs and variational autoencoders to capture the commonality of distinct games in a single latent representation. However, they generate game level snippets instead of playable game levels. In our work, we use the ability of Generative Adversarial Nets (GAN) to capture a low dimensional common representation of game levels from multiple games.

GANs first introduced by Goodfellow et al. (2014), are a way to learn generative models that reproduce examples from a training set by an adversarial process between a generator and a discriminator, both modeled by deep neural networks. The training process produces a generator that can take a random sample from low dimensional latent space and generate an example that is indistinguishable from the training set. This ability to generate more examples given a training set has been used in the PCG community to bootstrap training sets. Volz et al. (Volz et al. 2018) trained a GAN generator and explored the low dimensional latent space using an evolutionary search algorithm to find novel levels. Giacomello et al. (Giacomello, Lanzi, and Loiacono 2018; Giacomello, Lanzi, and Loiacono 2019) followed a similar approach of latent space exploration to generate novel DOOM levels. GANs, by definition, are trained to mimic training examples. We will show how variety in the training set impacts the novelty of the game levels generated.

One approach to increasing the training set size is to use GANs themselves to generate new training data. Torrado et al. (Torrado et al. 2019) have used a conditional embedding self-attention GAN (CESAGAN) to capture long distance dependencies in game levels. After each epoch of training, the generated playable levels are added to the training corpus to increase the number of examples. Park et al. (Park et al. 2019) used GANs to capture the patterns from a small set of examples to generate a larger batch of solvable training examples using multi-stage generation in the context of educational games. GANs capture what they see in the training examples, so working with a small training set might restrict the variety of game levels expressed by the generator. In this work, we propose a novel method to increase the training set to overcome this problem, which will be discussed in the next section.

In PCG as game levels are generated procedurally one needs fitness or evaluation metrics to determine the merit of each approach and identify desirable levels (Shaker, Togelius, and Nelson 2016; Shaker, Smith, and Yannakakis 2016). To simulate human evaluation, automated agents are used to play the games (Silva et al. 2018; Volz et al. 2018). Volz et al. evaluated level solvability using an agent to play the level, and they quantified difficulty based on the configuration of tiles in the level generated. We also use agents to play our levels to determine if a level is solvable. Novelty is typically defined as a distance measure between nearest neighbors (Lehman and Stanley 2011). In our work, we consider a level to be novel if it requires a unique sequence of actions to solve. We will define novelty based on distance between gameplay solutions as detailed in the later sections.

It is desirable to generate a solvable game level that is novel. It is also desirable that the generator can be controlled to generate levels that are interesting to play. Snodgrass and Ontañón (2016), in their work on PCG using multi-dimensional Markov chains, introduced constraints on the game elements in the levels generated, such as the existence of a specific number of difficult tile combinations. Khalifa et al. (Khalifa et al. 2019) evaluated game levels based on how an agent plays the game and the type of actions the agent performs like high jumps, long jumps, stomp kills, etc. Zhu et al. (Zhu, Wang, and Zyda 2018) evaluated the similarity between games based on a game event analysis of human's gameplay. In our generator, we also consider gameplay to evaluate our generated models. Snodgrass et al. (Snodgrass, Summerville, and Ontañón 2017) defined a plagiarism metric to see how much of the training levels was captured in the generated levels. We use a similar metric based on

gameplay to evaluate the amount of variety captured from training to generated levels.

## Approach

In this research we trained a GAN to generate game levels with the same gameplay in four distinct games, from a single random seed. We selected games that have similar game physics and game actions. In this section, we describe the game selection rationale, training set creation, and the GAN architecture.

### Games and Level Representation

The General Video Game Artificial Intelligence (GVGAI) framework and Video Game Description Language (VGDL) together provide a generic solution that can be used to represent and realize common 2D video games (Perez-Liebana et al. 2019). VGDL is a text-based description language that can be used to represent two-dimensional arcade games with grid-level physics. The language allows for the definition of individual sprites with custom properties including directional speed, interactions with other sprites, movement, scoring and determining termination conditions. The GVGAI framework provides a large set of predefined games in VGDL. The framework also provides agents that can play the games based on various heuristics. In this work we use both the framework to represent the games and the agents to test solvability of the generated game levels.

The set of four games selected from GVGAI (*Boulderdash, Link, Zelda* and *Roguelike*) follow grid physics and have similar actions available to the player. Interaction of the player's avatar with dynamic elements in the games like the monsters, moving tanks, and falling boulders, create distinction between the games. Falling boulders, which obey gravity is unique to *Boulderdash*. In *Roguelike* and *Link* there are solid walls with locked doors or breakable walls that require the player to first pick up a key or pickaxe to pass. In *Zelda* the layout of walls creates narrow pathways for the avatar to negotiate. These differences result in a variety of sprite patterns. One cannot just replace sprites in the level of one game with sprites from another game to generate levels in the other games. The ability of enemies in the games are also unique.

### Training Corpus Generation

As described above, the training corpus is a list of samples where each sample is a set of four game levels from *Boulderdash*, *Link*, *Zelda,* and *Roguelike*. The games were selected to have similar gameplay. By similar gameplay, we imply that following an equivalent action sequence in all the games will typically complete the level successfully. The training sets are created using an approach comparable to Shaker et al. (Shaker et al. 2015), who used abstract game timelines (sequence of actions in a game along with time deltas between actions) to generate game levels. The objective is to place obstacles that complement the actions at the right time and location such that the player action in the game timeline is necessary to move forward in a game.

As outlined in the algorithm (Algorithm 1), a training example creation starts with a set of grid points on an empty grid. The starting point is usually chosen on the top left quadrant and the goal point is chosen on the bottom right. A sequence of actions is selected that will take the player's avatar from one grid point to the next. This action sequence (e.g., jump, break a wall, pick a sword) is carried out through the grid as sprites like walls and locks are placed in the way to match the corresponding action. The action sequences are varied by changing the order of actions or permuting the combinations of actions randomly. Multiple combinations of actions that take the player's avatar from the start to the goal state are considered. The same action sequence is used in all the games considered, but the specific game's dynamics requires the placing of different obstacles to match the action. For example, in *Boulderdash* one has to avoid falling boulders and in *Roguelike* one needs to first pick up a key before passing a locked gate. The approach is generic and can be used to generate levels for multiple games starting from a common action sequence and path through the grid.

### Branched Generative Adversarial Network Model

We used deep convolutional GANs to model game levels in multiple games by using an innovative branched generator matched with individual game specific discriminators.

```
GenerateSolutionActionSequence (A: action set, G: grid dimensions )
  do
    Pick grid points →( a, b, c, d)
  while ( lines [ab, bc, cd] don't intersect)
  Set start position→ a
  Set final goal position → d
  Set intermediate goal positions →(b, c)
  Initialize solution path S := {}
  while (next position is not reached):
     Pick action act ∈ A based on a preset probability distribution
     validate
     if (pre-conditions for action is met)  and
       (adding act to S will keep avatar within G) and
       (avatar will progress to next position):
             add act to S
return S

GenerateTrainingLevel (S: solution action sequence, E: empty level,
                       a: start position, g: goal position)
  Set player avatar sprite → position a
  Set goal sprite → position g
  foreach act  in S
    add sprites to E such that act is the right action at that position
    move avatar position to match result of act
  return E
```

Algorithm 1: Training level generation algorithm.

A GAN typically consists of two types of deep networks: a single generator and a single discriminator. In our novel architecture we have constructed a generator that starts from a random seed like a typical GAN but branches into four different outputs as seen in Figure 2. Each of the outputs corresponds to game-levels in four different games in the training set. We have as many discriminators as there are branches in the generator. Each discriminator is tied to a single game and distinguishes between generated examples and training examples. The intuition is that the latent space and unbranched layers capture the commonality across the games while the branched layers capture the differences. Independent GANs, would not learn any common patterns as the only common element, the random input from the latent space, cannot be trained.

Each training sample is a multi-channel binary matrix, with each channel representing one type of sprite in the game and each grid point being a binary representation of the presence of the corresponding sprite at that grid point. The discriminators train independent of one another. Binary cross entropy loss from the discriminators is added to conditional loss from the generator. The generator loss is the sum of the binary cross entropy between the training sample and the generated image along with conditional loss if the number of sprites does not match the training level. The generators use batch normalization between convolution layers and LeakyReLU activation along with a final sigmoid activation to generate game level output. Each of the discriminators use a dropout of 30% to reduce overfitting.

The generator generates four grid physics games (*Boulderdash, Link, Zelda* and *Roguelike*) of size 16x16 from an initial input of 128 normally distributed random numbers. The training sample and generated game levels are represented as a tensor with nine channels, one each for each type of sprite (avatar, exit, floor, gold/health, key, lock, monster, wall and weapon). Unused channels are set to zero. The GANs were trained on a single GPU using 5000 examples in the training set. The training epochs ranged from 600 to 1200 with a batch size of 64.

## Evaluation

We use multiple evaluation metrics to quantify the quality of the generated levels. We check for solvability, the similarity of the gameplay between the different games, and novelty. Path similarity quantifies gameplay parity across games, while novelty measures variety in gameplay within levels of a single game. The details of the evaluation method are elaborated in this section.

### Solvability

Solvability is determined using automated agents available in the GVGAI framework. If a level can be solved at least once in 5 attempts by the automated agent, we consider the level to be solvable.

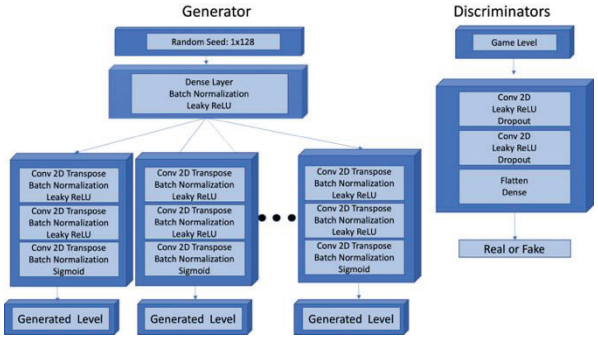


Figure 2: The GAN architecture consists of a branched generator and multiple parallel discriminators one for each game.

### Ideal Game Path Similarity

A shortest path is calculated from the avatar's initial position to the goal position with the stops along the way to pick up the necessary items to complete the level. The shortest path does not measure solvability because it does not take into account the dynamic aspects of the game. However, the shortest path is used to evaluate other metrics discussed in the subsequent sections. We determine an ideal path for the avatar in the level based on Dijkstra's shortest path algorithm (Dijkstra et al. 1959).

A path similarity measure is calculated between the shortest paths game levels of distinct games generated together. This path similarity distance is used to verify that the GAN model has captured the similarity between the games. The path similarity distance is the Manhattan distance between the grid locations in the path. The formula of the distance calculation is given by the formula,

$$d = \sum_{solution\ steps} |x_b - x_z| + |y_b - y_z|$$

Path similarity distance $d$ is given as a sum over all the steps in the solution, where $(x_b, y_b)$ and $(x_z, y_z)$ correspond to the grid position of the avatar in two distinct games, respectively. The shorter path is extended using copies of the goal location to match path lengths. For each generated set of games, the average path distance is calculated, between all pairwise combinations of games. Path similarity distance distribution across the sets of generated levels is compared to the path similarity distance distribution in the training set. If the GAN captures the gameplay similarity between distinct games, the distribution of path similarity distance should be the same between the training set and generated sets.

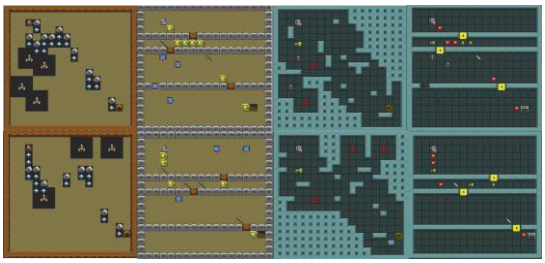

Figure 3: Two set of four games starting on the left Boulderdash, Link, Zelda and Roguelike from the training set.

## Novelty

The path similarity distance discussed in the earlier section was about similarity between distinct games, novelty is a measure of similarity within a game. Novelty is a binary relational property between two game levels of the same game. A level is novel with respect to another if the path taken by the avatar, represented by the sequence of actions is different. Thus, if a level requires a completely new sequence of actions to complete, then it would be considered novel. To evaluate novelty, we use the Levenshtein distance (Levenshtein 1966) between two ideal path action sequences. If the Levenshtein distance is large between two levels of a game, then we can claim that the gameplay will be different to a player. For example, if the solution action sequence for two Zelda levels is (right, right, pick key, up, right, right) and (right, up, pick key, up, right), the distance would be two as the number of edits to go from one sequence to the other is two.

## Results and Discussion

Figure 3 shows two sets of training game levels and Figure 4 shows four sets of generated game levels from the GAN generator. It is interesting to notice that the GAN learns to place boulders in *Boulderdash* above the diamonds. One can also see that in *Zelda*, the generator sometimes confines monsters behind walls. To get a glimpse into what the generator is really learning, we take two random latent vectors and their corresponding levels for one game. We then generate a third level from the vector sum of the first two latent vectors. Figure 5 shows an example from *Boulderdash*. We can see that the vector sum captures monster locations from the first level and some of the diamond locations from the second level into the third level. As expected, the GAN is encoding relative positional patterns of sprites from the training set into the latent space and is encoding the relative positions of sprites in the four different games into a single common latent representation.

We see from the examples in Figure 4 that the layout of sprites for monsters, keys, gold and health are similar across game levels in the different games. This correspondence is seen across all generated levels.

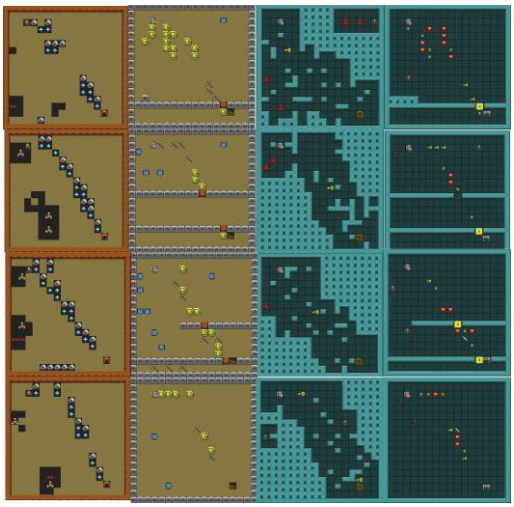

Figure 4: Four sets of generated levels using the GAN generator starting on the left Boulderdash, Link, Zelda, and Roguelike.

To validate and quantify the similarity of gameplay across games for generated game-level sets, we plot the average similarity distance between the ideal path for the avatar to reach the end state from the start state, picking up the necessary items and avoiding monsters. Figure 6 shows how the average path similarity distance is distributed in the baseline, training set and the generated set. The baseline represents sets of four game levels chosen randomly without considering gameplay similarity. The distance between the distributions can be quantified using the Wasserstein distribution distance. One can see that the generated sets of four levels have path similarity distribution closer (Wasserstein distance 161) to the training sets and further away from the baseline (Wasserstein distance 283). One could say the GAN has captured aspects of gameplay similarity across the four games.

Training sets are generated explicitly with the same action sequence for all four game levels resulting in similar gameplay for all four levels. The training levels have the gameplay flowing from the top left to the bottom right and one can see this captured by the GAN. GAN generator loss for the set of four games are averaged together in the gradient calculation with no other explicit constraint to match gameplay across games. The indirect constraint through loss results in some increase in gameplay variation in the generated levels as seen in Figure 6 as expected.

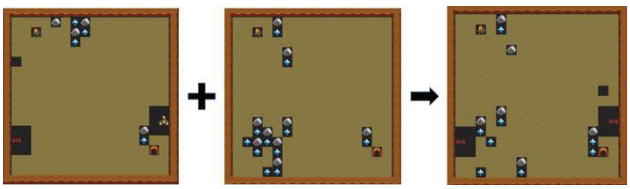

Figure 5: The third *Boulderdash* level generated from the vector sum of the latent seed vectors of the first two.

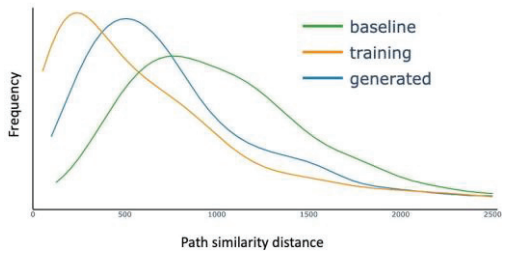


Figure 6: Distribution of average path similarity distance between distinct games in a training set and generated set.

We evaluate solvability of the generated levels by taking a set (50 levels for each game) of GAN generated levels and running an automated agent provided by the GVGAI framework. The agent is run up to five times to see if the level can be solved in allotted time of 2000 ticks. Figure 7 shows the relative solvability of the four different games. *Boulderdash* has a higher solvability (70%) over all the other games because it does not have any dependency between actions. For example, the need to pick up a key before being able to open a lock which is present in the other games. The reason *Zelda* has the lowest solvability among the generated levels (40%) might be due the need for the GAN to reproduce narrow paths between walls to go from the starting point to the goal and the need to have access to a key to finish the level.

The next evaluation metric we consider is novelty. Figure 8 shows how novelty of the generated levels compares with the novelty in the training levels of each game. We take 100 training and 100 generated levels for each game and calculate the distribution of pairwise Levenshtein distance between the ideal action sequence for the levels in the two sets. Distance is calculated between every level for a game with every other level in that set for that game. One can see that the variety or relative novelty of levels in the training set is captured by the GAN as the generated set has a similar distribution of values for the Levenshtein distance. By definition GANs are trying to mimic samples in the training sets, having a distribution of levels that are just as varied in the generated set as the original set implies that the generator has captured most of the complexity in the training set in its model.

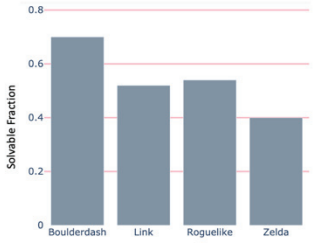


Figure 7: Generated levels solvable by an agent.

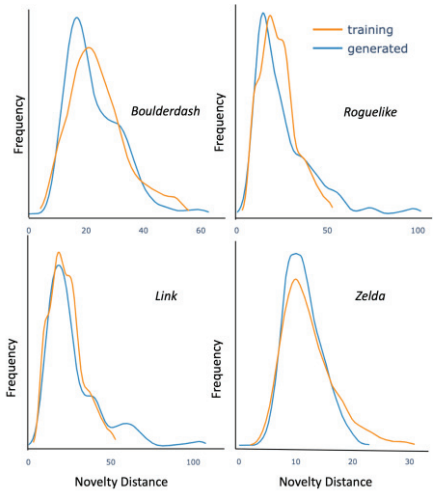


Figure 8: Pairwise Levenshtein distance distribution as a measure of novelty in generated levels vs training levels.

## Conclusion

GAN-based PCG often focuses on generating levels in individual games. There is an underlying commonality in how many games operate despite apparent variability on initial inspection, and we have shown that one can build a game-independent representation to capture that commonality of multiple games. Commonalities captured in a low dimensional latent space can then be explored to generate new interesting game levels, and possibly, new games. In this work we trained a novel branched GAN that can take a single random seed vector to generate parallel levels in four distinct games with similar gameplay, while capturing the variability seen in the training levels.

We have presented a novel approach to building the training corpus starting from a prescribed gameplay action sequence. By using gameplay, one is starting from a player's experience of the game. Building the training corpus based on the gameplay action sequence also guarantees that the paired levels generated have similar gameplay characteristics. We are currently using a simple rule-based algorithm but using answer-set programming techniques similar to the work done by Smith and Mateas (2011) could make this process more extensible.

A promising direction for future work is to formalize specific game characteristics of a broad selection of games that can be represented in a single common representation. It will also be instructive to determine if there are classes of games that can be grouped based on specific aspects of gameplay and game rules. Further, exploring the possibility of capturing the temporal elements of games in the latent space may lead to more powerful PCG frameworks for broad classes of games.

## References


Bentley, G. R., and Osborn, J. C. 2019. The videogame affordances corpus. In *Proceedings of AIIDE Workshop on Experimental AI in Games*.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1(1):269–271.

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.

Giacomello, E.; Lanzi, P. L.; and Loiacono, D. 2019. Searching the latent space of a generative adversarial network to generate Doom levels. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

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.

Guzdial, M., and Riedl, M. 2016. Learning to blend computer game levels. In *Proceedings of the Seventh International Conference on Computational Creativity*.

Guzdial, M., and Riedl, M. 2018a. Automated game design via conceptual expansion. In *Proceedings of Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Guzdial, M. J., and Riedl, M. O. 2018b. Combinatorial creativity for procedural content generation via machine learning. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Guzdial, M.; Reno, J.; Chen, J.; Smith, G.; and Riedl, M. 2018. Explainable PCGML via game design patterns. In *Proceedings of AIIDE Workshop on Experimental AI in Games*.

Guzdial, M.; Liao, N.; and Riedl, M. 2018. Co-creative level design via machine learning. In *Proceedings of AIIDE Workshop on Experimental AI in Games.*

Justesen, N.; Bontrager, P.; Togelius, J.; and Risi, S. 2019. Deep learning for video game playing. *IEEE Transactions on Games* 12(1):1-20.

Khalifa, A.; Green, M. C.; Barros, G.; and Togelius, J. 2019. Intentional computational level design. In *Proceedings of The Genetic and Evolutionary Computation Conference*, 796–803.

Lehman, J., and Stanley, K. O. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* 19(2):189–223.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, 707–710.

Park, K.; Mott, B. W.; Min, W.; Boyer, K. E.; Wiebe, E. N.; and Lester, J. C. 2019. Generating educational game levels with multistep deep convolutional generative adversarial networks. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2019. General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games* 11(3):195–214.

Sarkar, A., and Cooper, S. 2018. Blending levels from different games using LSTMs. In *Proceedings of AIIDE Workshop on Experimental AI in Games*.

Sarkar, A.; Yang, Z.; and Cooper, S. 2019. Controllable level blending between games using variational autoencoders. In *Proceedings of AIIDE Workshop on Experimental AI in Games*.

Shaker, M.; Shaker, N.; Togelius, J.; and Abou-Zleikha, M. 2015. A progressive approach to content generation. In *European Conference on the Applications of Evolutionary Computation*, 381–393. Springer.

Shaker, N.; Smith, G.; and Yannakakis, G. N. 2016. Evaluating content generators. In *Procedural Content Generation in Games*. Springer. 215–224.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games*. Springer.

Silva, F. D. M.; Borovikov, I.; Kolen, J.; Aghdaie, N.; and Zaman, K. 2018. Exploring gameplay with AI agents. In *Proceedings of Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference.*

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.

Snodgrass, S., and Ontañón, S. 2016a. An approach to domain transfer in procedural content generation of two- dimensional videogame levels. In *Proceedings of Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Snodgrass, S., and Ontañón, S. 2016b. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *IJCAI*, 780–786.

Snodgrass, S.; Summerville, A.; and Ontañón, S. 2017. Studying the effects of training data on machine learning-based procedural content generation. In *Proceedings of Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Summerville, A., and Mateas, M. 2016. Super Mario as a string: Platformer level generation via LSTMs. In *Proceedings of the First International Joint Conference of DiGRA and FDG*.

Summerville, A.; Guzdial, M.; Mateas, M.; and Riedl, M. O. 2016. Learning player tailored content from observation: Platformer level generation from video traces using LSTMs. In *Proceedings of Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

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.

Torrado, R. R.; Khalifa, A.; Green, M. C.; Justesen, N.; Risi, S.; and Togelius, J. 2019. Bootstrapping conditional GANs for video game level generation. *arXiv preprint arXiv:1910.01603*.

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

Zhu, T.; Wang, B.; and Zyda, M. 2018. Exploring the similarity between game events for game level analysis and generation. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, 1–7.