# Bootstrapping Conditional GANs for Video Game Level Generation

Ruben Rodriguez Torrado
*Game Innovation Lab*
*New York University - OriGen.AI*
Brooklyn, USA
rubentorrado@origen.ai

Ahmed Khalifa
*Game Innovation Lab*
*New York University*
Brooklyn, USA
ahmed@akhalifa.com

Michael Cerny Green
*Game Innovation Lab*
*New York University-OriGen.AI*
Brooklyn, USA
mike.green@nyu.edu

Niels Justesen
*IT University of Copenhagen*
København, Denmark
njustesen@gmail.com

Sebastian Risi
*IT University of Copenhagen*
København, Denmark
sebastian.risi@gmail.com

Julian Togelius
*Game Innovation Lab*
*New York University - OriGen.AI*
Brooklyn, USA
julian@togelius.com

***Abstract*—Generative Adversarial Networks (GANs) have shown impressive results for image generation. However, GANs face challenges in generating contents with certain types of constraints, such as game levels. Specifically, it is difficult to generate levels that have aesthetic appeal and are playable at the same time. Additionally, because training data usually is limited, it is challenging to generate unique levels with current GANs. In this paper, we propose a new GAN architecture named *Conditional Embedding Self-Attention Generative Adversarial Network* (CESAGAN) and a new bootstrapping training procedure. The CESAGAN is a modification of the self-attention GAN that incorporates an embedding feature vector input to condition the training of the discriminator and generator. This allows the network to model non-local dependency between game objects, and to count objects. Additionally, to reduce the number of levels necessary to train the GAN, we propose a bootstrapping mechanism in which playable generated levels are added to the training set. The results demonstrate that the new approach does not only generate a larger number of levels that are playable but also generates fewer duplicate levels compared to a standard GAN.**

***Index Terms*—Generative Adversarial Networks, Conditional Embedding, Self-Attention, Bootstrapping, General Video Game Framework, Functional Content Generation, Procedural Content Generation**

## I. Introduction

Procedural Content Generation (PCG) is a term defining methods in which content for games or simulations is created using programmatic means. Procedural Content Generation via Machine Learning (PCGML) is a PCG approach in which a machine learning component is trained on existing content to generate new game content [1] such as platform levels [2], strategy game maps [3], and collectible cards [4]. Different types of machine learning algorithms can be used, as long as they learn some aspect of the distribution of the training set in such a way that new content can be sampled from the model. Machine learning algorithms for PCGML include statistical methods such as n-grams [5], Markov chains [6], recurrent LSTM networks [2], and convolutional networks [7].

Research on PCGML is only a few years old, spurred on by the recent advancements in machine learning models for creative or generative tasks such as generating faces [8], music [9], or text [10] with impressive results. At first glance, it seems reasonable that the same methods could be used to generate content such as levels, characters, or quests for your favorite game. However, there are some crucial differences and challenges when dealing with game content.

The first difference is data scarcity. When training a machine learning model to create lifelike faces, coherent news stories or harmonious music, training data is abound. It is easy to find thousands of training examples, and deep learning models in particular achieve increasingly better results with more training data. However, for most games, only a limited amount of content exists. Super Mario Bros (Nintendo, 1985) has a few dozen levels, Mass Effect (BioWare, 2007) probably less than a hundred named characters, Skyrim ( Bethesda, 2011) only tens of non-trivial quests, and Grand Theft Auto V (Rockstar, 2013) a handful of car models and weapon types. Only relatively few games have user-made content of sufficient quantity and quality to make for a good training set, and this content is often not publicly available (e.g. the level corpus from Super Mario Maker).

The second difference is that many types of game content (in particular *necessary content* [11]) have functional requirements. A picture of a face where one eye is smudged out is still recognizably as a face, and a sentence can be agrammatical and misspelled but still readable; these types of content do not need to *function*. In contrast, game levels, in which it is impossible to find the key to the exit are simply unplayable, and it does not matter how aesthetically pleasing they are. The same holds true for a ruleset, which does not specify how characters move, or a car where the wheels do not touch the ground. In this respect, it is useful to think of most game content as being more like program code than like images. The problem is

978-1-7281-4533-4/20/$31.00 ©2020 IEEE



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

that most generative representations are not intrinsically well-suited to create content with functional requirements. Many such functional requirements depend on counting items or non-local relation and these are hard to capture with standard network architectures.

One way of addressing the problem of functional requirements is to combine PCGML with a search-based approach, exploring the space learned by a trained model rather than just sampling from it. In particular, *Latent Variable Evolution*, originally invented to produce fingerprints able to bypass authentication schemes [12], was successfully used to generate Super Mario Bros level segments with functional properties [7]. However, it is desirable to have a model learn the functional requirements, regardless of whether we later choose to generate through random sampling or search.

This paper introduces a new PCGML method that seeks to incorporate functional requirements while at the same time mitigating training data scarcity. At the core of this method is a new GAN architecture, the CESAGAN, which incorporates self-attention to capture nonlocal spatial relationships and a conditional input vector. In order to help the GAN learn functional requirements, we input to the network not only the raw level geometry but also aggregates of level features. Levels are generated through sampling the generator network, and all generated levels are tested for playability. Levels that are playable, and sufficiently different from the levels in the training set, are added to the training set for continued training of the GAN. This way, the training set is bootstrapped from a very small number of levels to a much larger set.

## II. Related Work

This section discusses a general overview of procedural content generation (PCG), followed descriptions of applications of GAN-based level generation in video games.

### A. Procedural content generation

PCG refers to the use of computer algorithms to produce content. These techniques have played an important role in video games since the early eighties, with such examples as Rogue (Glenn Wichman, 1980), Elite (David Braben and Ian Bell, 1984), and Beneath the Apple Manor (Don Worth, 1978). An important reason for the early popularity of PCG in games is its the ability to produce large amounts of content with a negligible memory cost, fitting on a small floppy disk [13]. Important recent games employing PCG include Spelunky (Derek Yu, 2008), The Binding of Isaac (Edmund McMillen and Florian Himsl, 2011), and No Mans Sky (Hello Games, 2016).

Designers and developers are using PCG [14] for a variety of different purposes, such as tailoring game contents to the player's taste, assisting creative content generation, reducing the time and cost for designing and developing games, exploring new types of games, or understanding the design space of games. PCG can be used to generate any type of content from textures to game rules. Some types are easier than the others, such as generating vegetation (trees, bushes etc) in games (Interactive Data Visualization Inc, 2000) [15]. While level generation might seem like a trivial problem, as it has been used since the early days of video games, this is not the case. Most known level generation algorithms are tailored to generating content for a particular game [16, 17] using a significant amount of game or genre specific knowledge to make sure the generated levels are playable and enjoyable.

Recently, there has been an increase in the development of generalized PCG algorithms that can work on multiple games [18]. Search-based Procedural Content Generation (SB-PCG) [11] methods use search algorithms to generate levels, applying simulations and automated playthroughs to validate the generated content. Procedural Content Generation via Machine Learning (PCG-ML) [1] methods use (small) example sets of levels to train on, after which they generate new levels. This paper proposes a new PCG-ML approach as a solution to the small training set used in PCG-ML methods allowing the algorithms to learn better models for level generation.

### B. Generative Adversarial Networks

The architecture of a Generative Adversarial Network (GAN) [19] can be understood as an adversarial game between a generator ($G$), which maps a latent random noise vector to a generated sample, and a discriminator ($D$), which classifies generated samples as real or fake. These adversaries are trained simultaneously, striving towards reaching a state where the discriminator maximizes its ability to classify correctly and the generator learns to create new samples that are good enough to be classified as genuine. GANs became popular in recent years due to their impressive results in tasks such as image generation. However, training GANs is not a trivial procedure: the training process is often unstable, where the generator produces unrealistic samples, or the discriminator is no more accurate than a coin toss. For these reasons, many extensions have been proposed to improve the training process and the quality of the results. For example, Mirza and Osindero [20] feed a vector $y$ to train $G$ and $D$ conditioned to generate descriptive tags which are not part of training labels. In addition, Bellemare et al. [21] proposed a new training methodology to grow both the generator and discriminator architecture complexity progressively, reaching a higher-quality on the CELEBA dataset.

More recently, high-quality results have been reported using attention mechanisms in deep learning [22]. Attention mechanisms are a very simple idea that identifies the most relevant variables dynamically in more complex deep neural network architecture such as, convolutional neural network (CNN). Recently, Zhang et al. [23] combined attention mechanism and GANs to generate and discriminate high-resolution details as a function of only spatially local points in lower-resolution feature maps.

In this paper, we propose a new GANs architecture which combines both ideas, conditional GANs and attention mechanisms, in other words, we combine attention mechanisms with conditional GANs in order to improve the quality and diversity of generated levels.



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

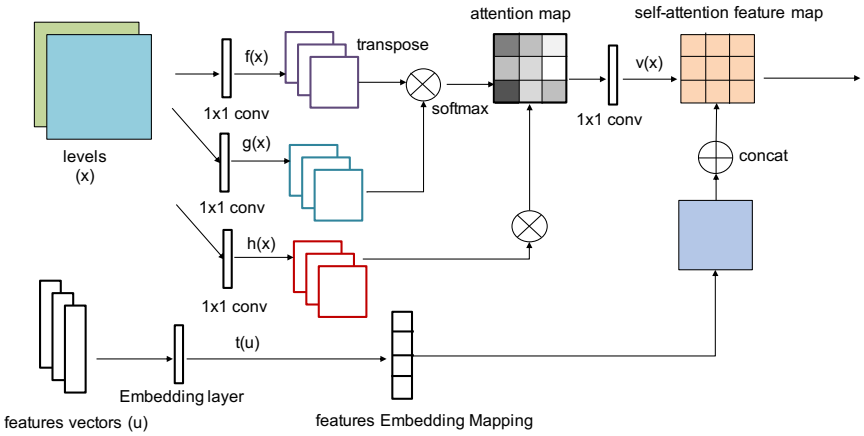


Fig. 1: The architecture for our Conditional Embedding Self-Attention Generative Adversarial Network (CESAGAN). The approach combines a SAGAN architecture (top), with a conditional embedding for the feature information vector $u$ (bottom). We concatenate the feature embedding mapping and the self-attention feature map to combine SAGAN with the conditional vector representation $u$. This network is applied to both the generator (G) and discriminator (D)

### C. PCG and GANs

Volz et al. [7] and Giacomello et al. [24] first introduced the idea of unsupervised learning techniques for PCG. Giacomello et al. [24] trained a GAN to create plain level images for DOOM by combining image and topological features extracted from human-designed content. Though this methodology generates realistic levels from the topological point of view, the playability of the generated levels was not tested.

Volz et al. [7] combined GANs with latent variable evolution (LVE) [25] to optimize the input latent vector of a GAN generator to create levels for Mario Bros. Deep Convolutional GANs (DCGANs) were adapted to generate levels, and CMA-ES searched in the space of latent vectors. The results demonstrated that it is often possible to generate both realistic and playable levels for Mario Bros video game. In this paper, we are comparing our method with the approach introduced by Volz et al. [7].

Hu et al. [26] tackled the problem of generating content that follows certain constraints by introducing the constraints as a part of the loss function using a regularization method. This approach helps the trained model to battle the problem of the functional content but doesn't help to increase the diversity of the generated content. In contrast to Hu et al, our new work introduces constraints as part of the network architecture in the conditional embedding layer to overcome the functional content challenge.

### D. General Video Game AI Framework

The General Video Game Artificial Intelligence framework [27] (GVG-AI) is a framework built to run 2D arcade-like games written in Video Game Description Language (VGDL) [28]. Originally developed for game-playing competition, GVG-AI has since evolved to be a primary faucet for a variety of research projects and competitions [27]. These competitions and projects have resulted in the creation of more than 200 controllers and well over 120 games.

## III. Methodology

GANs have proven very successful in generating images and similar types of content that do not have structural and functional representation. However, when generating game levels, simple GAN approaches have several shortcomings, which the method presented in this paper tries to overcome:

1) Reducing the amount of information necessary to train the discriminator; most games have just a few human-designed levels available to form a training set.
2) Increasing quality; GANs often generate levels with low quality that are sometimes unplayable.
3) Increasing diversity; the diversity and number of unique generated levels are limited with previous approaches [7].

We approach challenge 1 with a bootstrapping technique and challenge 2 with a new GAN architecture. Both of these techniques also make strides towards Challenge 3.

### A. Conditional Embedding Self-Attention Generative Adversarial Networks (CESAGANs)

Previous GAN-based models for level generation are built using convolutional layers. A convolutional layer is a local operation whose correlation depends on the spatial size of the kernel. For example, in a convolution operation for level generation, it is hard for an output on the top-left position to have any correlation to the output at bottom-right. A deep convolution network with many layers would be required which will increase the large search space.

This phenomena has become larger for video games level generation where just three tiles/pixels located far away from each other (e.g. avatar-door-key) must be correlated to generate a playable level. An intuitive solution to this problem could



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

be reduce the kernels sizes and layers located deeper in the network to be able to capture this relationship later. However, this approach would increase the number of layers of the deep neural network significantly and thus make the GAN training more unstable [29, 30].

One potential method that could keep balance between efficiency and capturing long-range dependencies is a self-attention GAN (SAGAN). A Self-attention GAN [31] is based on three different vectors: query (f), key (g) and value (h) which are three different mappings (e.g. output of a single perceptron neural network) of the input data (e.g. an image) 1. The query and key undergo a matrix multiplication then pass through a softmax, which converts the resulting vector into probability distribution attention map. This attention map determines the weight of each of the tiles and keep it in memory. Finally, the attention map is multiplied by the value to determine the relationship at a position in a sequence by attending to all positions within the same sequence.

In our experiments, we adapt self-attention GANs [23] for video game level generation. The mechanism is shown in Figure 1. The one-hot tile level representation from the hidden layer is transformed into two feature spaces $f$ and $g$ to compute the attention, where $f(x) = W_f x$ and $g(x) = W_g x$ are the query and key. We transpose the query and matrix-multiply it by the key, $s_{i,j} = f(x_i)^T g(x_j)$ and take the softmax on all the rows in order to calculate the attention map:

$$\beta_{j,i} = \frac{exp(s_{i,j})}{\sum_{i=1}^{N} exp(s_{i,j})} \tag{1}$$

As we described above, $\beta_{j,i}$ indicates the correlation at a position $i$ when mapping the area $j$. Finally, the output of the attention layer is $o_j = v(\sum_{i=1}^{N} \beta_{j,i} h(x_i))$, where $v$ and $h$ are the output of the 1x1 convolutional future. This self-attention map layer helps the network capture the fine details from even distant parts of the image and creates a memory for future correlations.

In SAGAN, the attention module has been used to train the generator and the discriminator, minimizing the hinge version of the adversarial loss [23]:

$$\begin{aligned} L_D = &- \mathbb{E}_{(x,y)p_{data}}[min(0, -1 + D(x, y))] \\ &- \mathbb{E}_{(z),p_x(y),p_{data}}[min(0, -1 + D(G(z), y))] \end{aligned} \tag{2}$$

$$L_G = -\mathbb{E}_{(z),p_x(y),p_{data}} D(G(z), y), \tag{3}$$

where $z$ is the latent vector. However, this architecture does not guarantee that the generated levels respect different playability-required features such as a minimum/maximum numbers of different type of objects.

For that reason, we extend SAGANs to train the generator and discriminator conditioned to an auxiliary information input feature vector $u$ of each level. As vector $u$ we are using the the count of each unique tile of the target levels. The mapping representation of the feature vector $u$ into the self-attention map is learned by a neural network (the embedding network) during the supervised training process of SAGAN. The embedding network transforms the vector $u$ into a new feature continuous space (an embedding representation) $t(u)$, where $t(u) = W_t u$.

Embedding representations reduce memory usage and speed up neural network training when compared with more traditional representations of auxiliary information (feature vector $u$) such as a one-hot encoding [32]. In addition, the new representation of $u$ in the embedding space enables the correlation of similar values of categorical variables. Such correlations are more difficult to capture with a simple one-hot representation. This allows the new representation to find more general patterns of the feature vector and therefore allows the SAGAN architecture to generalize better.

The output of the embedding mapping $t(u)$ is concatenated with the output of the attention layer $o(i)$, conditioning the adversarial loss functions 2 and 3 to the input feature vector $u$:

$$\begin{aligned} L_D = &- \mathbb{E}_{(x,y),p_{data}}[min(0, -1 + D((x, y)|\mathbf{u}))] \\ &- \mathbb{E}_{(z),p_x(y),p_{data}}[min(0, -1 + D((G(z), y)|\mathbf{u}))] \end{aligned} \tag{4}$$

Finally, The CESAGAN network uses 1×1 convolutions in the discriminator and 1×1 deconvolutions or transposed convolution layer in the generator. Additionally, we employ batchnorm both in the generator and discriminator after each layer and ReLU activations. For the conditional embedding layer, we use a simple fully connected layer that is concatenated with the self-attention feature map. Each type of tile is encoded with an ASCII character in the textual representation of the level and uniquely mapped to a numerical identity.

$$L_G = -\mathbb{E}_{(z),p_x(y),p_{data}} D((G(z), y)|\mathbf{u}) \tag{5}$$

We call the proposed method Conditional-Embedding Self-Attention Generative Adversarial Networks (CESAGAN) (Figure 1). It is important to note that the additional conditioning input of CESAGAN enables the production of levels using specific input information such as number of enemies and player avatars. In other words, the new architecture gives significantly more control to generate game content such as levels with desired characteristics and extends the application areas PCG can be applied to.

### *B. Bootstrapping*

Our new architecture CESAGAN has the potential to improve the playability of generated maps. However, we still require a considerable number of levels to train the discriminator in order achieve diversity in the generated output. For that reason, we have proposed a bootstrapping mechanism to improve the efficiency of CESAGAN architecture explained above. This bootstrapping mechanism takes advantage of built-in game properties that are shared with other types of computer programs, particularly the fact that they can be checked for functionality by attempting to play/execute them. This differentiates game levels from other domains, like pure images.



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

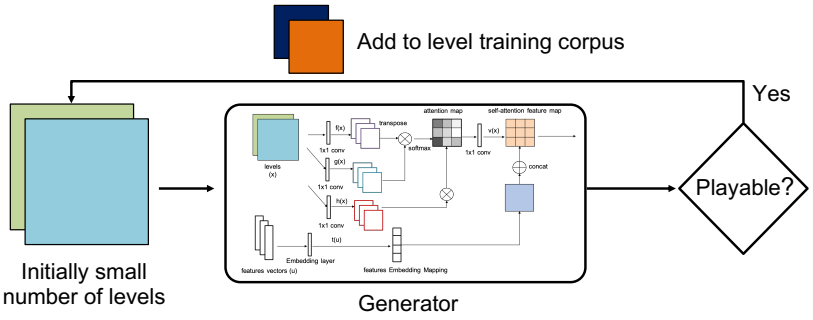


Fig. 2: Conditional Embedding Conditional Self-Attention Generative Adversarial Network (CESAGAN) with bootstrapping. The bootstrapping mechanism increases the number of training examples after passing a playability and diversity test. Bootstrapping improves the quality of the GAN's discriminator.

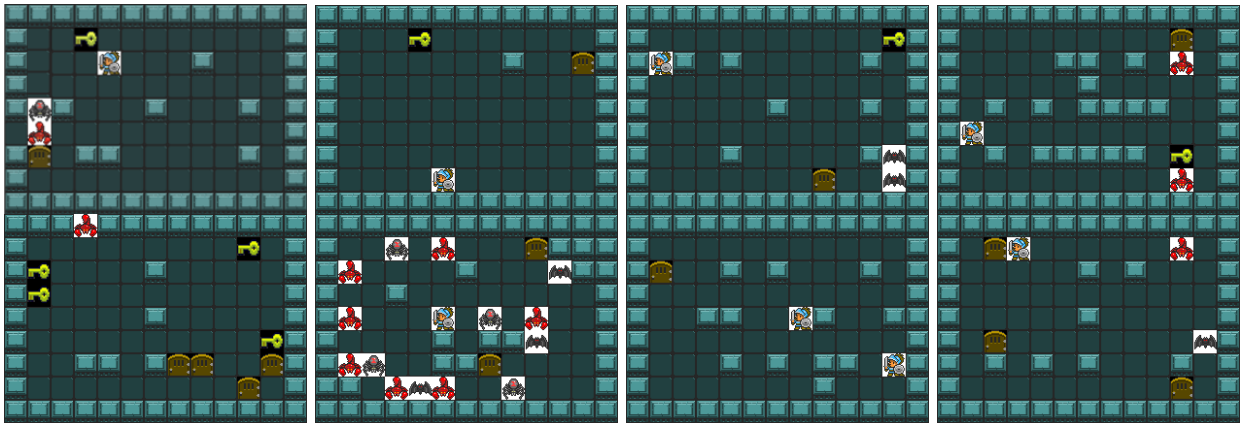

Fig. 3: Examples of generated levels by random sampling the trained CESAGAN with bootstrapping. The top four levels are levels that staisfy the constraints while the lower one break them.

After each epoch, a new set of levels is generated, on which a playability and duplicate analysis are carried out to identify unique playable levels. We propose a set of heuristics for the playability test which is detailed in the Experiments section. In order to identify duplicates levels, we propose the following workflow inspired by Valladao et al. [33]:

1) We project all the levels generated in a 2-D spaces using PCA; we denominate the new 2D space as $\Omega$
2) We run k-means algorithm to divide our 2-D generated levels in k clusters. In order to identify an optimum number of clusters (k), we use Elbow method. The elbow method run the k-means algorithm with different values of k (for instance between 1 to 1,000 clusters) and picks the k that minimize the intra-cluster variation.
3) Finally, the level which is closer to the centroid of each cluster is selected as representative level obtaining the set of non duplicated levels $\Omega^{'} \subset \Omega$.

After this workflow, $\Omega^{'}$ is added to the initial set of training data, transforming $p_{data}$ to $p^{'}_{data}$ for training the generator and the discriminator for the next epoch (see Figure 2). This technique doesn't check duplicates between the generated levels and the current training data set. This might cause some duplicates between the new added levels and the current training data which we decided to investigate in future work.

## IV. Experiments

Generator and discriminator are trained with RMSprop with a batch size of 32 and the default learning rate of 0.0001 for 10,000 iterations. To train the discriminator we used two sets of training levels. The first one consists of 45 human-designed levels, including the five levels that come with the GVGAI framework, while the second one only consists of the five human-designed levels from the GVGAI framework. Figure 4 shows some examples of the training data, where the top five images are the levels that come with the GVGAI framework.

| Tile type | Symbol | Identity |
|---|---|---|
| Wall | w | 0 |
| Empty | . | 1 |
| Key | + | 2 |
| Exit door | g | 3 |
| Enemy 1 | 1 | 4 |
| Enemy 2 | 2 | 5 |
| Enemy 3 | 3 | 6 |
| Player | A | 7 |

TABLE I: Mapping of Zelda encoding tiles. Each symbol is encoded as a on-hot encoding for the GAN.



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

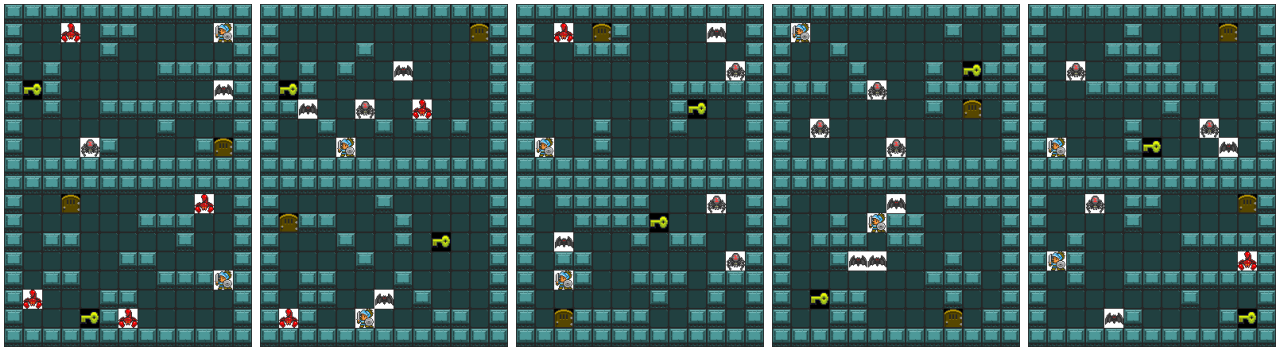

Fig. 4: Example of human designed levels used for training. The 5 levels shown at the top are the ones that come with the GVGAI framework.

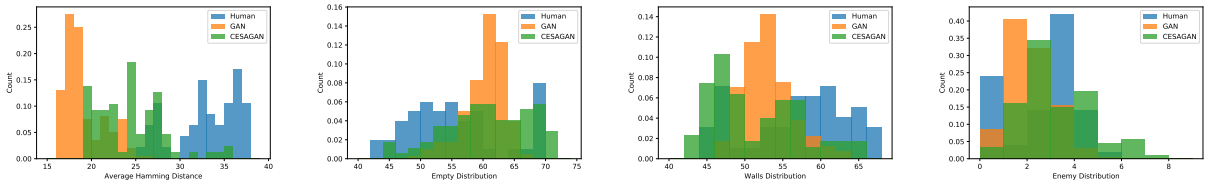


Fig. 5: Distribution of the average hamming distance and different game tiles for CESAGAN, GAN, and Human Levels.

To evaluate the presented approach we use the game Zelda from the GVGAI environment [34]. This game is a VGDL port of the dungeon system from "The Legend of Zelda" (Nintendo, 1986). In this game, the player needs to collect a key and reach the exit door without getting killed by the moving enemies. The player can kill the enemies using their sword for extra points. Table I shows the encoding for the tiles in Zelda. Our baseline is the adaptation of GAN architecture proposed by Volz et al. [7] for Zelda (now adapted to Mario levels). In order to compare both approaches we generated 15,000 levels for both models.

For the bootstrapping playability check, the following seven heuristics are used; they are based on our knowledge about the game to ensure playability (Figure 4):

- There is only one player avatar.
- There is only one key.
- There is only one door.
- Enemies cover less than 60% of the empty space (it is harder to beat the level when there are too many enemies)
- The avatar can reach the key using an A* algorithm.
- The avatar can reach the door using an A* algorithm.
- The level has a border of walls to prevent the avatar and enemies to go outside the level.

For the conditional vector $u$, we are only using the number of player, key, and door tiles described in table I. These tiles are selected not only because of their relevance to game playability: the addition of other tiles might restrict the generator's possibility space and will not allow it to find new interesting levels.

For more generality, one could use automated game playing agents. For example, our system could have used a planning agent from the GVGAI framework [27] to check for playability, rather than heuristics. However, doing so would have increased the amount of time that it takes to train, as the agents have to play each level.

| Model | Results with 45 training levels | |
|---|---|---|
| | Playable levels | Duplicated levels |
| Baseline GAN | 19.4% | 39.4% |
| CESAGAN | 58% | 37.6% |

TABLE II: Comparing Ratios of playable and duplicated levels with a training set of 45 levels w/o bootstrapping. The ratio are calculated based on 15,000 levels from the generator.

## V. Results

In order to evaluate the efficiency of the GAN models, generated levels are tested for playability and duplication for CESAGAN with and without bootstrapping. They are compared with the state-of-art techniques below. We define two sets of training data: (1) five human-designed levels and (2) 45 human-designed levels. For the first set, these five levels are the levels that comes with the original GVGAI framework. The extra 40 levels are later designed by students and added to the corpus.

Table II shows the results for CESAGAN without any bootstrapping using a total of 45 Zelda levels as training data. The CESAGANs architecture improves upon both metrics against state-of-art techniques. These results show the potential of the new architecture to improve the playability of generated artifacts. However, the number of duplicate levels has the same order of magnitude with respect to the state-of-art.

Training on 45 levels is likely not realistic for the majority of video games; just a few human levels are usually avail-



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

| Model | Results with 5 training levels | |
|---|---|---|
| | Playable levels | Duplicated levels |
| Baseline GAN | 24.6% | 98% |
| CESAGAN | 37% | 88% |
| CESAGAN + boostrapping | 42% | 57% |

TABLE III: The ratios of playable and duplicated levels using five levels in the training set with/without bootstrapping. The ratio are calculated based on 15,000 levels from the generator.

able to train for most games. For this reason, CESAGAN with bootstrapping is trained on just five human-designed levels. The results (Table III) demonstrate that the approach increases the percentage of playable levels, while reducing the number of duplicates considerably. In addition, if we compare CESAGAN and CESAGAN with bootstrapping we observe that bootstrapping reduce dramatically the number of levels duplicated and also help towards increasing playability. Finally, these results show how both mechanism benefit each other, in other words, bootstrapping mechanism improves the number of playable levels getting by CESAGAN operator alone. By looking on both tables (Table II and III), we find that the percentage of playable levels for Baseline GANs drops when the number of training set increase. We think this is due to the big decrease of the number of duplicated levels (from 98% to 39.4%) as having higher percentage duplicated levels increase the chance of having higher percentage of duplicated playable levels.

Figure 3 shows eight levels generated by random sampling the trained CESAGAN with bootstrapping. The top row shows four different playable levels [1] which not only satisfy all the constraints but also have small amounts of enemies similar to the human designed levels even though the constraint only restricted enemy cover to less than 60% of the empty tiles. The bottom row displays four unplayable levels that do not follow different constraints but the most notable broken constraints are the numerical constraints (number of avatar, doors, keys, and enemies).

For further analysis of the generated content, we calculate the average hamming distance by Norouzi et al. [35] between the playable levels for the different techniques and compare against the 45 human designed levels. Each level is compared to all the other levels in the same set and the average hamming distance is calculated (number of different tiles). Figure 5 shows the average hamming distance between levels from the same set. Human designed levels have the highest distance (33.31 and stdev 3.87), followed by CESAGAN levels (24.34 and stdev 3.7), then GAN generated levels (19.12 and stdev 2.3). CESAGAN produces a more diverse set of levels that are as different from each other as possible compared to traditional GANs.

Distributions of number of different tiles in the playable levels generated CESAGAN and GAN are also calcualted and compared to the distributions in the human levels. Figure 5 shows the different distributions of empty, wall, and enemy tiles in order. Avatar, Key, or Door tile distributions are not shown because they are always equal to 1 in playable levels based on the defined constrained. CESAGAN levels have nearly double the standard deviation than GAN levels , meaning the CESAGAN model has the ability to generate more diverse levels than the normal GAN. Having a higher standard deviation is a double edged weapon, however. On one hand, it enables new innovative levels that never been seen. On the other, it deviates from the input style from the dataset.

[1] https://drive.google.com/file/d/1LmzyaEi4Kj4AKsIlBKrEOTvsm-3DuwJl/view?usp=sharing

## VI. Conclusion

We introduce a new GAN architecture – Conditional Embedding Self-Attention Generative Adversarial Network (CESAGAN) with bootstrapping mechanism – for video game level generation. The results of the experiments confirm the original concern that the state-of-art in GAN has limitations when applied to procedural content generation (PCG). In particular, GANs have difficulty in generating playable and unique levels when few training samples are available. To address this challenge, we introduce Conditional Embedding Self-Attention Generative Adversarial Network (CESAGAN) with bootstrapping. This new architecture is a modification of SAGAN, with an additional feature conditional vector to train the discriminator and generator. The results show a considerable improvement in playability and diversity for 15,000 generated levels with respect to the state-of-art.

One of the next challenges for CESAGAN with bootstrapping is to train on more complex video games such as Boulderdash or train with more complex architectures in place of the conditional feature. In addition, using a deep neural network to select the most relevant levels for bootstrapping could decrease the number of duplicate levels even further.

## Acknowledgments

Ruben Rodriguez Torrado/Michael Cerny Green acknowledge the support of OriGen.AI to publish this work. Ahmed Khalifa acknowledges the financial support from NSF grant (Award number 1717324 - "RI: Small: General Intelligence through Algorithm Invention and Selection."). Michael Cerny Green acknowledges the financial support of the SOE Fellowship from NYU Tandon School of Engineering. All authors acknowledge Per Josefsen and Nicola Zaltron, who were responsible for the 45 human-designed levels.

## References

[1] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *Transactions on Games*, vol. 10, no. 3, 2018.

[2] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," in *FDG*, 2016.

[3] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *Twelfth Artificial Intelli-*



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.

*gence and Interactive Digital Entertainment Conference*, 2016.

[4] A. J. Summerville and M. Mateas, "Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning," in *AIIDE*, 2016.

[5] S. Dahlskog, J. Togelius, and M. J. Nelson, "Linear levels through n-grams," in *International Academic MindTrek Conference*. ACM, 2014.

[6] S. Snodgrass and S. Ontanón, "Controllable procedural content generation via constrained multi-dimensional markov chain sampling." in *IJCAI*, 2016.

[7] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Genetic and Evolutionary Computation Conference*. ACM, 2018.

[8] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[9] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *Workshop on neural networks for signal processing*. IEEE, 2002.

[10] A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever, "Better language models and their implications," 2018.

[11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, 2011.

[12] P. Bontrager, A. Roy, J. Togelius, N. Memon, and A. Ross, "Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution," in *International Conference on Biometrics Theory, Applications and Systems*. IEEE, 2019.

[13] I. Karth, "Elite (1984)," https://procedural-generation.tumblr.com/post/112509130817/elite-1984-elite-created-by-ian-bell-and-david, 2015.

[14] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.

[15] I. D. V. Inc, "Speedtree," https://store.speedtree.com/, 2000.

[16] J. Taylor and I. Parberry, "Procedural generation of sokoban levels," in *International Conference on Intelligent Games and Simulation*, 2011.

[17] L. Ferreira and C. Toledo, "A search-based approach for generating angry birds levels," in *Computational Intelligence and Games*. IEEE, 2014.

[18] A. Khalifa and M. Fayek, "Automatic puzzle level generation: A general approach using a description language," in *Computational Creativity and Games Workshop*, 2015.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014.

[20] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[21] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, jun 2013.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017.

[23] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[24] E. Giacomello, P. L. Lanzi, and D. Loiacono, "Doom level generation using generative adversarial networks," in *Games, Entertainment, Media Conference*. IEEE, 2018.

[25] P. Bontrager, J. Togelius, and N. Memon, "Deepmasterprint: Generating fingerprints for presentation attacks," *arXiv preprint arXiv:1705.07386*, 2017.

[26] Z. Hu, Z. Yang, R. R. Salakhutdinov, L. Qin, X. Liang, H. Dong, and E. P. Xing, "Deep generative models with learnable knowledge constraints," in *Advances in Neural Information Processing Systems*, 2018.

[27] D. Perez, J. Liu, A. Abdel Samea Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *Transactions on Games*, 2019.

[28] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," *Dagstuhl Reports*, 2013.

[29] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016.

[30] N. Kodali, J. Abernethy, J. Hays, and Z. Kira, "On convergence and stability of gans," *arXiv preprint arXiv:1705.07215*, 2017.

[31] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.

[32] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[33] D. M. Valladao, R. R. Torrado, B. Flach, S. Embid *et al.*, "On the stochastic response surface methodology for the determination of the development plan of an oil & gas field," in *SPE Middle East Intelligent Energy Conference and Exhibition*. Society of Petroleum Engineers, 2013.

[34] R. D. Gaina, A. Couetoux, D. J. N. J. Soemers, M. H. M. Winands, T. Vodopivec, F. Kirchge$\beta$ner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player GVGAI competition," *Transactions on Computational Intelligence and AI in Games*, 2017.

[35] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in neural information processing systems*, 2012.



Authorized licensed use limited to: Universitaetsbibliothek der RWTH Aachen. Downloaded on March 02,2022 at 23:34:24 UTC from IEEE Xplore. Restrictions apply.