Dungeon level generation using generative adversarial network: an experimental study for top-down view games

Daniele Fernandes e Silva[1,2], **Rafael Piccin Torchelsen**[1], **Marilton Sanchotene de Aguiar**[1]

[1]Federal University of Pelotas (UFPel)
Pelotas, RS – Brazil

[2]Federal Institute of Education, Science and Technology Farroupilha
Alegrete, RS – Brazil

daniele.fernandes@iffarroupilha.edu.br

{rafael.torchelsen,marilton}@inf.ufpel.edu.br

Abstract. *Manual designing levels for games is a complex task, often demanding time and effort from the game designer. An option for this is using algorithms to generate such levels, improving its scalability. Currently, such procedural content generation methods can be guided by hand-crafted rules or, as in more recent approaches, by learning from existing data. In deep learning, generative adversarial networks have been proven to effectively generate samples for a given distribution in an unsupervised manner. Unfortunately, the training and usage of GANs show many challenges, such as convergence problems, instability, and mode collapse. In addition to these problems, we show that learning to generate valid levels is not trivial. In this paper, we demonstrate an empirical analysis of the use of GANs, pointing out the best state-of-the-art practices and highlighting future directions in this research field.*

1. Introduction

In the digital game field, Procedural Content Generation (PCG) is used to reduce, in some cases even eliminate, the manual workload for designing game aspects such as levels, vegetation, music, texture, and others [Viana and dos Santos 2021]. However, the quality of PCG results heavily depends on the game designer's ability to parameterize and map constraints inside the algorithm. The level design consists of elements and rules to dictate the gameplay, such as terrain layout, number of enemies and collectibles, and others. We can procedurally create these elements with some restrictions to avoid generating unplayable or unbeatable levels. For dungeon games, the variability of levels is essential to improving gameplay.

Considering the success of Generative Adversarial Networks (GAN) in the image processing field, with impressive results in image-to-image translation [Zhu et al. 2017], image-to-text translation [Zhang et al. 2016], image blending [Wu et al. 2017]; some works have explored it for procedural generation, known as Procedural Content Generation via Machine Learning (PCGML) [Summerville et al. 2018, Liu et al. 2021, Chen and Lyu 2022, Gutierrez and Schrum 2020]. Unfortunately, such machine learning algorithms are data hungry [Adadi 2021], and generating sample cases of valid levels

for dungeon games is challenging. Our experiments compare the performance of GANs trained with small datasets.

This paper presents an experimental study of dungeon-level generation using different GANs architectures trained on a small amount of synthesized data from a dungeon game. We devise a playability metric for comparing the number of valid levels generated by each GAN model, highlighting the problems of each architecture and pointing out directions for improving results. The main contributions of this work are:

- New PCG approach to dungeon games in the style of maps from the Diablo series;
- Devising a gameplay validation measurement;
- Achieving better results for the placement of game elements by the generated map;
- Evaluation between different GAN architectures for level generation.

2. Generative Adversarial Networks

Generative Adversarial Networks consist of two neural networks (a *generator* and a *discriminator*) competing against each other [Goodfellow et al. 2014, Goodfellow et al. 2020]. This architecture aims to learn a probabilistic distribution that best represents the given/target dataset.

Many architectures and loss functions were proposed in the literature to stabilize the training process and improve model convergence. For instance, the replacement of fully connected networks, known as VanillaGAN, by deep convolutional networks [Radford et al. 2015], and the use of loss functions based on Jensen–Shannon divergence [Goodfellow et al. 2014, Goodfellow et al. 2020] or Wasserstein divergence [Gulrajani et al. 2017].

GANs are already used in popular games, such as The Legends of Zelda [Torrado et al. 2020, Zhang et al. 2020] and Doom [Giacomello et al. 2018]. We aim to generate levels for more challenging scenarios: terrains greater than the previous ones, with large open areas (very similar to those encountered in Diablo games [Barman et al. 2018]).

2.1. Training with small dataset

Creating a vast dataset of dungeon levels is challenging and time-consuming for game designers, as such levels usually must follow restricted game constraints and need a diversity of level generations. By using deep learning algorithms, we aim to simplify this task: just giving some samples of valid levels for a model to learn how to generate many other valid levels.

However, training GANs with small datasets is challenging, as it quickly leads training to overfit and, for GANs, to mode collapse. Many works have used data augmentation to mitigate this issue [Ziviani et al. 2022], but few of them are applied to tiled data level [Ping and Dingli 2020]. In this work, we make use of traditional data augmentations, as well as a custom-develop transformation tailored for top-down levels. Section 3.2 describes the augmentations used in our experiments.

2.2. Regularization for GANs

Regularization techniques are widely used to stabilize GANs' training. Most known regularization techniques include the use of batch normalization or dropout lay-

ers [Radford et al. 2015], gradient penalty [Gulrajani et al. 2017], and spectral normalization [Miyato et al. 2018]. We analyze the performance of these techniques in the scope of dungeon-level generation.

3. Methodology

This experimental study aims to verify the applicability of GANs for a dungeon-level generation. We conduct our experiments in two steps of the pipeline shown in Figure 1. We can notice the GAN architecture on top, with the generator and discriminator, while the playability validation can be seen at the bottom. The validation is used as a criterion to measure the generation potential during the training process, which we discuss in the following sections.

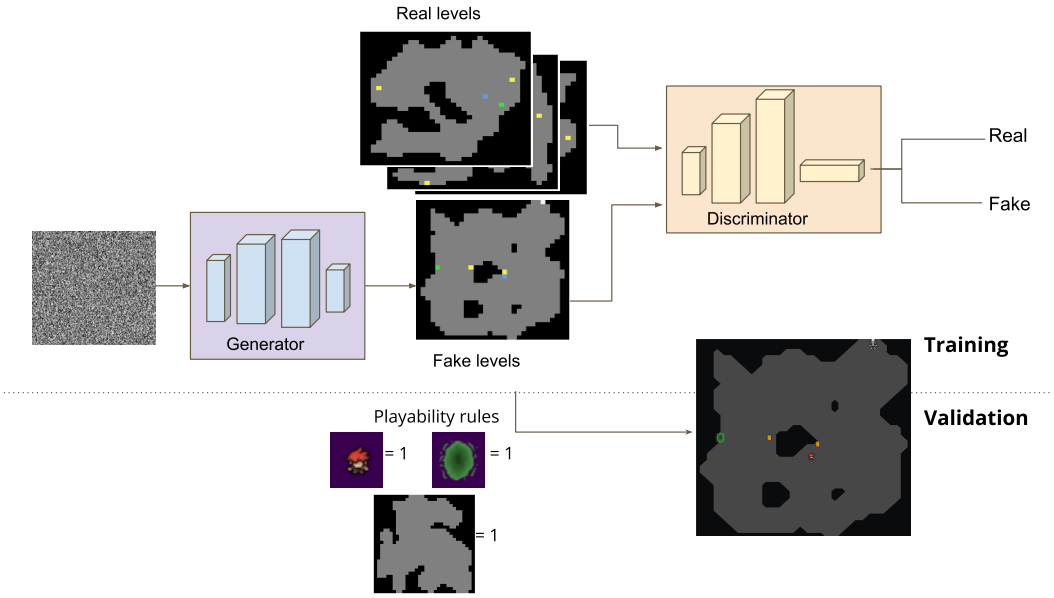

Figure 1. Pipeline used in the proposed study. It performs training and evaluation according to playability rules.

3.1. Dungeon Game

The [Padilha 2022] dungeon game used for this work procedurally generates the entire level terrain (wall and floor) and arrangement of elements, such as access (entrance and exit), enemies, and collectibles positions. We consider a level valid if it meets the following requirements:

1. all levels must have only one portal tile (exit);
2. all levels must have only one player position tile (start position);
3. all floor tiles must be accessible to the player; and,
4. each tile can be only one of floor, wall, player, portal, enemy, coin.

3.2. Data preparation

The data for training the GANs were synthesized from the dungeon game mentioned in the previous subsection. The dataset was composed of 200 levels of size 32×32. Figure 2 show some synthesized samples.

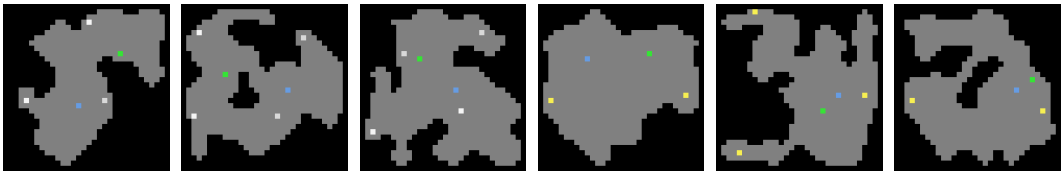

Figure 2. Samples synthetized with [Padilha 2022] work. Such samples are included in our training dataset.

These dungeon levels are represented in a grid format comprising seven different tiles. The values assigned to the grid are distinct integers that represent the respective tile and have a visual representation according to the detailed color mapping in Table 1.

Table 1. Game elements with corresponding colors for the generated levels.

Data	Value	Color	Data	Value	Color
Floor	0		Skeleton	4	
Wall	1		Cannon	5	
Player	2		Coin	6	
Portal	3				

After the data acquisition step, these data were encoded through one-hot encoding and applied some data-augmentation techniques. Following traditional data augmentation transformations used on level generation [Ping and Dingli 2020], we have applied vertical and horizontal flips and 90º rotations. We have also tailored a binary dilation as data augmentation to increase the expressiveness of the dataset.

3.3. Architectures

Many GAN architectures exist in the literature; however, only some works explore their applications for level generation in dungeon games.

3.3.1. Vanilla GAN

The architecture proposed by [Goodfellow et al. 2014], composed of fully connected layers (Table 2), is adjusted according to the loss function described in Equation 1.

$$V(G,D) = \mathbb{E}_{p_{data}(x))} log D(x) + \mathbb{E}_{p_g(x)} log(1 - D(G(z)) \quad (1)$$

Table 2. Implementation of the VanillaGAN architecture based on [Goodfellow et al. 2014].

Layer	Discriminator		Generator	
	Type	in_features, out_features	Type	in_features, out_features
1	Linear-LeakyReLU	(7168,512)	Linear-ReLU	(30,256)
2	Linear-LeakyReLU	(512,256)	Linear-BN-ReLU	(256,1024)
3	Linear-Sigmoid	(256,1)	Linear-Tanh	(1024, 7168)

The discriminator tries to maximize the loss function by learning to classify the output x into real – following the function $D(x)$ – and the generated data $G(z)$ into fake

– following the function $D(G(z))$, where z refers to the latent space as input to the generator. The generator acts directly on the second term of the equation to minimize the loss function $(1 - D(G(z)))$ maximizing the output of $G(z)$; thus, the generator tries to trick the discriminator into labeling the generated data as real.

We used VanillaGAN as a baseline for our experiments, evaluating whether fully connected networks can learn the data distribution of dungeon levels. The generator of this architecture use *tanh* in the output layer, and the SGD optimizer (Stochastic Gradient Descent), being the *learning rate* defined in 0.025 and *momentum* 0.5.

3.3.2. Deep Convolution GAN

Convolutional architectures are computationally less expensive based on fully-connected layers, popularly used in problems involving images [Gu et al. 2018, Li et al. 2021]. We perform experiments using conv layers, followed by Bach Normalization (BN) and upsamples layers. The detailed architecture descriptions can be seen in Table 3.

Table 3. Implementation of DCGAN architecture based on [Radford et al. 2015].

Layer	Discriminator		Generator	
	Type	**n_filters, k_size, stride**	**Type**	**n_filters, k_size, stride**
1	Conv-LeakyReLU-Dropout	(7,16), (3,3), (2,2)	Conv-BN-ReLU-Upsample	(20,64), (3,3), (1,1)
2	Conv-LeakyReLU-Dropout-BN	(16,32), (3,3), (2,2)	Conv-BN-ReLU-Upsample	(64,32), (3,3), (1,1)
3	Conv-LeakyReLU-Dropout-BN	(32,64), (3,3), (2,2)	Conv-Tahn	(32,7) (3,3), (1,1)
4	Linear-Sigmoid	(1024,1)		

Based on the DCGAN architecture [Radford et al. 2015], we adapted the input layer of the generator to a matrix structure, which receives a noise matrix as input. We follow the same hyperparameters of [Radford et al. 2015]: Adam optimizer with *learning rate* 0.0002 and β_1 = 0.5.

3.3.3. Wasserstein GAN with gradient penalty

[Arjovsky et al. 2017] presents a modification in the loss function to improve the fit of the predictive model. Based on the Wasserstein distance, the Wasserstein GAN (WGAN) architecture is defined according to Equation 2, where $\tilde{x} = G(z)$ refers to the data generated from a latent space z.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \mathbb{E}_{x\ \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x}\ \mathbb{P}_g}[D(\tilde{x})] \tag{2}$$

The authors use the term critic to name the discriminator, as they do not train the critic to classify but to favor the generator's adjustments. Like VanillaGAN, the critic tries to maximize the distance between the real data distributions of the generated data. In contrast, the generator attempts to approximate these distances with generations closer to the real data distribution $D(\tilde{x})$.

The WGAN with Gradient Penalty (WGANGP) architecture included this penalty in the original WGAN function and the penalty coefficient defined by λ (Equation 3). The

generator is adjusted n critical iterations during the training process for each epoch iteration. In this way, it is possible to train the critic to its optimal model without presenting the mode collapse problem.

$$L = \mathbb{E}_{\tilde{x}\ \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\ \mathbb{P}_r}[D(x)] + \underbrace{\lambda \mathbb{E}_{\hat{x}\ \mathbb{P}_{\hat{x}}}[(\parallel \nabla_{\hat{x}} D(\hat{x}) \parallel_2 -1)^2]}_{\text{penalty}} \tag{3}$$

We perform experiments with this approach to compare the use of gradient penalty as a regularization method on tiled dungeon data. The WGANGP implementation uses the same DCGAN architecture [Radford et al. 2015]. The hyper-parameters were the same defined by [Gulrajani et al. 2017]: Adam optimizer with *learning rate* 0.0001, $\beta_1 = 0$ and $\beta_2 = 0.9$, penalty factor $\lambda = 10$ and number of iterations of the critic $n_{\text{critic}} = 5$. We trained this network with variations of the *learning rate* value for a better network adjustment.

3.4. Metric evaluation

Evaluating the performance of generative models is not trivial, and we have found no metric tailored for this experimental case. Based on [Torrado et al. 2020], we have devised a playability metric that quantitatively measures valid levels in a subset of GAN's output. A given level must match the specifications defined according to Equation 4 to be considered valid.

$$M_e = \begin{cases} \sum M_e(i,j) = 1; e \in 2,3 \\ \\ f(M_0) = 1 \end{cases} \tag{4}$$

where M is the level matrix composed of e game elements (channels). The first term of the equation is related to rules 1 and 2, while the second term refers to rule 3 from Section 3.1. The function $f(M_0)$ counts the number of 4-neighbor-connected floor tiles (we expected a single connected component). In other words, such metric measures how many valid levels each model can generate.

4. Experimental Results

We performed the experiments on a hardware device with a Linux Mint 20 Cinnamon operating system, a Core i5 processor with 16 GB of RAM, and an NVIDIA Titan Xp GPU with 12 GB of VRAM. In addition, we performed a quantitative analysis for each experiment, training the neural network for 10,000 epochs and evaluating each epoch with the proposed metric with a population of 1,000 samples. We have picked the epoch with the best metric for each GAN architecture in all training.

4.1. Comparison of architectures

VanilaGAN was set as the baseline for the experiments. According to our playability metric, the predictive model produced only 4.8% of valid levels when training the network for 360 epochs. However, the quality of the levels did not present variability due to the generated samples having one big blob of floor tiles (with no internal walls), besides the disposition of other elements being majoritarian near walls, as shown in Figure 3.

After 3,000 training epochs, the model stopped learning, stagnating in the same generation (Figure 4). The traditional configuration of the VanillaGAN cannot learn the behavior of the target data, and it is very susceptible to the problem of mode collapse.

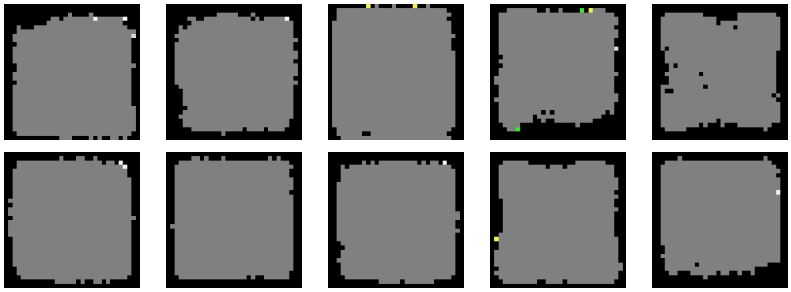

Figure 3. Level generated by VanillaGAN architecture. We can see few variability, with one large terrain blob, no corridors and elements placed close to the wall.

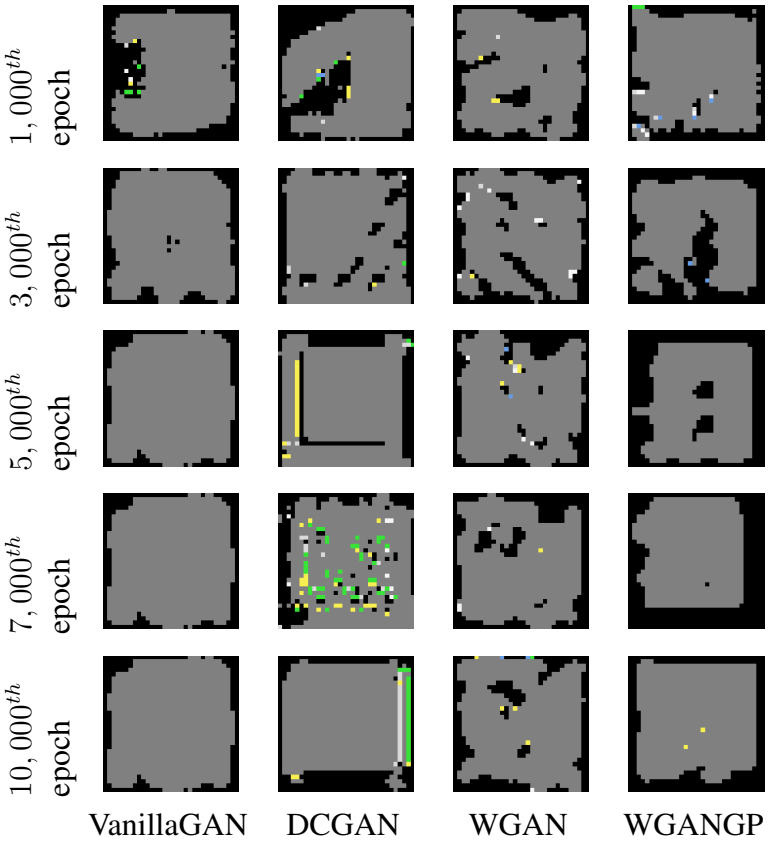

Figure 4. Visualization of generator outputs in each training epoch on VanillaGAN, DCGAN, WGAN and WGANGP architectures.

We tested the architectures DCGAN, WGAN, and WGANGP for the following experiments, all using the same convolutional architecture. We adjust the critic iteration (WGAN) and lambda (WGANGP) to improve the output results. In our experiments, there was no significant difference between the Adam and RMSprop optimizers; therefore, we prefer using the second due to reduced hyperparameters.

We obtained a longer training time with convolutional networks than fully connected networks. The WGAN architecture achieved 7.4% of valid levels with 7,466 training epochs and remained stable over 10,000 epochs. However, DCGAN and WGANGP showed instability when trained for over 3,000 epochs. These models had also overfit to a

behavior similar to VanillaGAN results when achieving 4,147 and 5,626 epochs, respectively.

Analyzing the training up to 4,000 epochs, DCGAN produced 8.7% valid levels by training the model for 594 epochs, and WGANGP obtained 8.6% valid samples with 504 training epochs. In addition, we observed that dungeon-level training using convolutional networks could better learn the distribution of terrain, with slightly more varied configurations of internal walls, as shown in Figure 5.

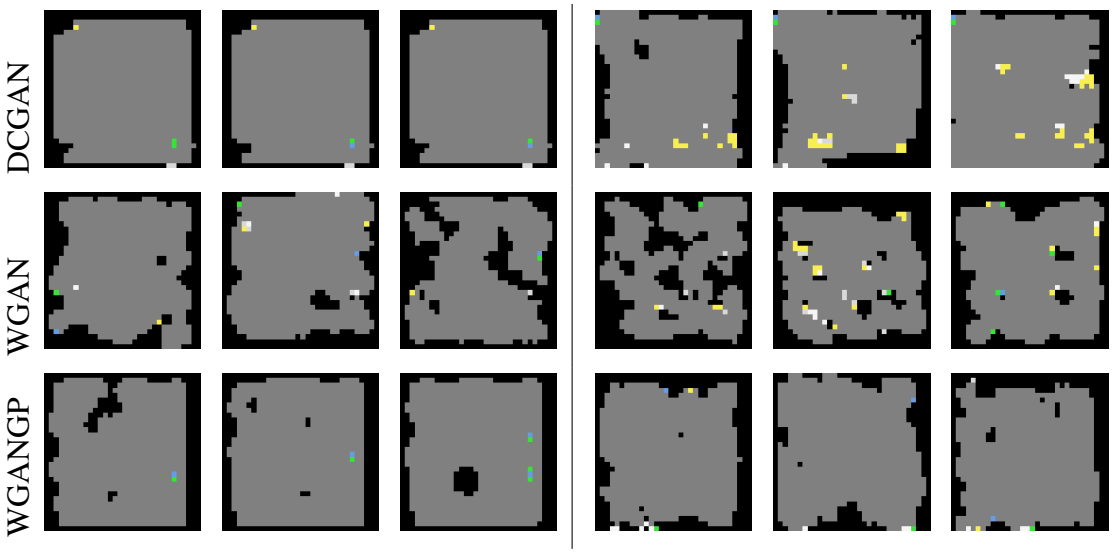

Figure 5. Level generated by each convolutional network architecture. We can see some corridors in generations using DCGAN and WGAN architectures and few variabilities in the terrain layout of generations with WGANGP architecture.

4.2. Comparison of regularization techniques

Traditional convolutional architectures were originally introduced using Batch Normalization layers in Discriminator/Critic network. [Miyato et al. 2018] propose a technique to stabilize the training process in GANs called Spectral Normalization (SN). We evaluated the impact of replacing BNs for SNs.

The use of SN layers resulted in a delay in the convergence of the predictive models. For example, the architectures DCGAN-SN, WGAN-SN, and WGANGP-SN took 9,407, 3,555, and 4,185 epochs to reach the values shown in Table 4.

Table 4. Playability comparison of different regularization methods for each architecture with 10,000 epochs.

Architecture	Regularization	Valid level (%)
DCGAN	BatchNormalization	(mode collapse)
	SpectralNormalization	**13.9**
WGAN	BatchNormalization	7.4
	SpectralNormalization	**11.4**
WGANGP	BatchNormalization	(mode collapse)
	SpectralNormalization	**2.9**

Even being a less optimized approach, the results of the generations became better visually. In an empirical evaluation, the levels are similar to the target data. We observed that GANs' architectures with SN are less susceptible to mode collapse (when compared to using BNs), as shown in Figure 6. Another issue found in BN architecture is that the elements tended to remain close to the wall. The samples generated from the training of the GAN with SN had the elements more sparsely arranged across the terrain. We believe that the SN better handles elements distribution, i.e., making channels with less occurrences (player, portal, enemies, and coins) having equal importance as the floor and wall channels.

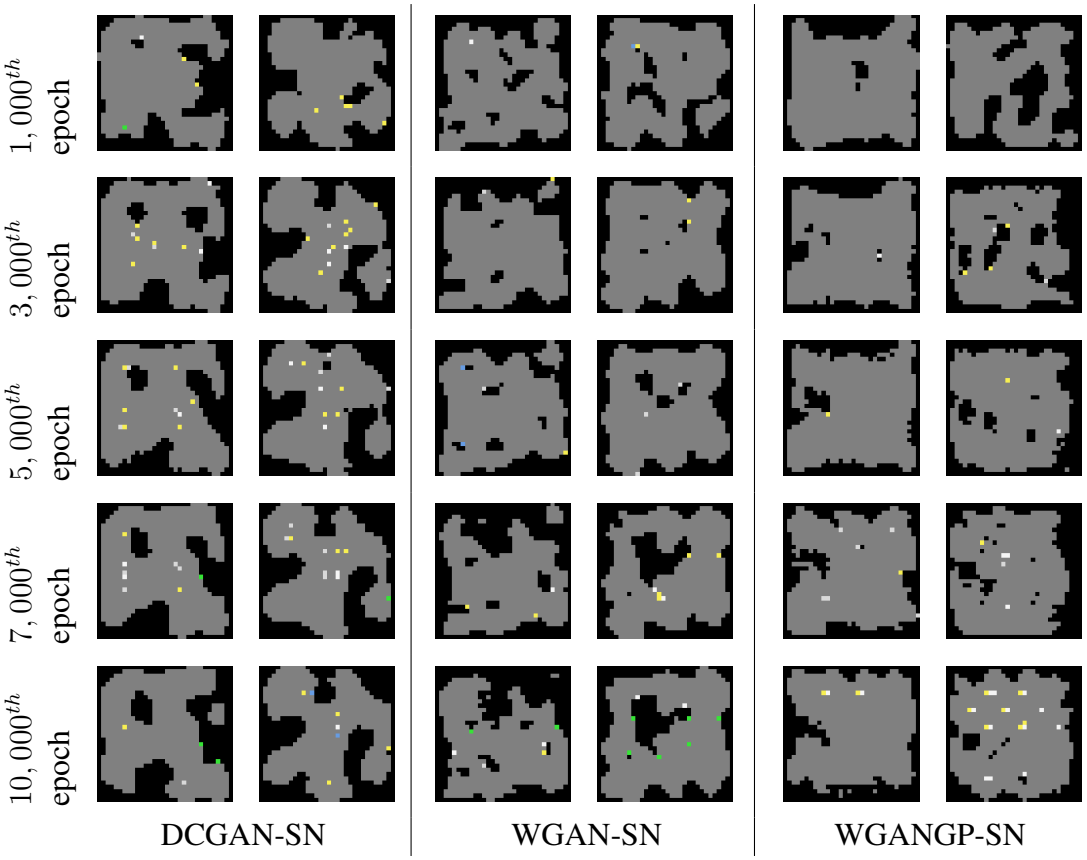

Figure 6. Visualization of generator outputs in each training epoch on VanillaGAN-SN, DCGAN-SN, WGAN-SN, and WGANGP-SN architectures. We can see better results in terrain layout and placement of elements.

5. Discussion

The experiments allowed observing the performance of GANs trained with different layers, loss functions, and regularization techniques. We adopted a playability metric to measure the performance, representing the percentage of valid levels generated in a subset. The training of different convolutional architectures and loss functions had little impact on improving level generation. On the other hand, using Spectral Normalization dramatically enhances the playability metric of GANs' results.

As mentioned, all GANs based on fully-connected layers got poor results. An explanation for this is that fully-connected layers are much more prone to overfitting than

convolutional layers, especially in the scenario of small datasets. Another advantage of using convolutional networks is the flexibility of using the generator for creating variable-size levels just by using latent vectors of different resolutions. We experimented with the models trained with SN in the task for generating 64×64 levels. The architectures achieve a performance of 0.4%, 0.9%, and 0.1%, respectively, in DCGAN-SN, WGAN-SN, and WGANGP-SN. Figure 7 shows the results of this experiment.

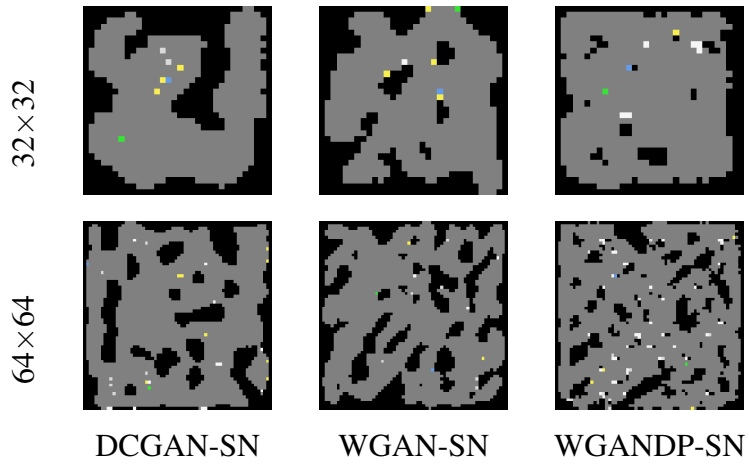

Figure 7. Level generated in different resolutions of each convolutional network architecture using SN.

Despite the small value of our playability metrics, all evaluated models quickly learn to generate valid terrains; in less than 1,000 epochs, they reach values between 80% and 90% for valid terrains. Empirically, we observed from the VanillaGAN generations that the terrains tend to be wider (Figure 3), while in the convolutional ones, it is possible to observe some occurrences of corridors (Figure 5). Placing the elements properly was more challenging than creating the terrain topology of the levels.

In this training, the networks had a lot of difficulty placing elements (player, enemies, coins) on the terrain. VanillaGAN and GANs with convolutional architectures suffered from the problem of elements being arranged near walls. We assume that the difficulty of learning the disposition of the elements is due to an imbalance between terrain (floor and walls) and elements (player, portal, enemies, and coins), which can be minimized through regularization techniques.

6. Conclusion and Future Work

This work presented different approaches based on GANs for generating top-down dungeon levels. We demonstrate the entire pipeline of data synthesis and organization, data augmentation, training in different architectures, and validation of the generated results.

The experiments allowed us to observe the learning ability of the approaches for generating levels with few data and some classic problems of GANs, such as the difficulty of convergence, mode collapse, and training instability when trained with traditional architecture. We mitigate these problems using the SN technique, allowing an improvement in the convergence of the predictive model, such as overcoming overfitting. Furthermore, empirically we noticed a greater output similarity with the target data from this approach, demonstrating the potential for generating new samples within the same domain.

As a limitation of this work, there is still a minimal result of valid levels. Therefore, we intend to explore new ways of tuning the architecture and using data augmentation techniques to increase model convergence. To validate the levels, it was necessary to implement heuristics that met the game elements' disposition rules. Therefore, we used no metric to evaluate the variability of levels generated in the experiments. In future work, we intend to evaluate diversity based on the work of [Choi et al. 2020], using Frechet's inception distance (FID) and Learned Perceptual Image Patch Similarity (LPIPS) metrics.

Acknowledgements

We thank the reviewers for their insightful comments and NVIDIA Corporation for donating the GeForce Titan Xp GPU used for this research. Furthermore, this study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

Adadi, A. (2021). A survey on data-efficient algorithms in big data era. *Journal of Big Data*, 8(1):24.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan.

Barman, N., Zadtootaghaj, S., Schmidt, S., Martini, M. G., and Möller, S. (2018). Gamingvideoset: a dataset for gaming video streaming applications. In *2018 16th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6. IEEE.

Chen, Z. and Lyu, D. (2022). Procedural generation of virtual pavilions via a deep convolutional generative adversarial network. *Computer Animation and Virtual Worlds*, 33(3-4):e2063.

Choi, Y., Uh, Y., Yoo, J., and Ha, J.-W. (2020). Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197.

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

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA. MIT Press.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans.

Gutierrez, J. and Schrum, J. (2020). Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*.

Liu, J., Snodgrass, S., Khalifa, A., Risi, S., Yannakakis, G. N., and Togelius, J. (2021). Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Padilha, R. F. (2022). *Analisando o engajamento de jogadores através de mapas procedurais*. Trabalho de conclusão de curso (bacharelado em ciência da computação), Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas.

Ping, K. and Dingli, L. (2020). Conditional convolutional generative adversarial networks based interactive procedural game map generation. In *Future of Information and Communication Conference*, pages 400–419. Springer.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

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. (2020). Bootstrapping conditional gans for video game level generation. In *2020 IEEE Conference on Games (CoG)*, pages 41–48. IEEE.

Viana, B. M. F. and dos Santos, S. R. (2021). Procedural dungeon generation: A survey. *Journal on Interactive Systems*, 12(1):83–101.

Wu, H., Zheng, S., Zhang, J., and Huang, K. (2017). Gp-gan: Towards realistic high-resolution image blending.

Zhang, H., Fontaine, M., Hoover, A., Togelius, J., Dilkina, B., and Nikolaidis, S. (2020). Video game level repair via mixed integer linear programming. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 151–158.

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. (2016). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks.

Ziviani, H. E., Chávez, G. C., and Silva, M. C. (2022). Applying a conditional gan for bone suppression in chest radiography images. In *Anais do XLIX Seminário Integrado de Software e Hardware*, pages 25–36. SBC.