Proceedings of the Twentieth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2024)

Controllable Game Level Generation: Assessing the Effect of Negative Examples in GAN Models

Mahsa Bazzaz, Seth Cooper

Khoury College of Computer Sciences, Northeastern University
bazzaz.ma@northeastern.edu, se.cooper@northeastern.edu

Abstract

Generative Adversarial Networks (GANs) are unsupervised models designed to learn and replicate a target distribution. The vanilla versions of these models can be extended to more controllable models. Conditional Generative Adversarial Networks (CGANs) extend vanilla GANs by conditioning both the generator and discriminator on some additional information (labels). Controllable models based on complementary learning, such as Rumi-GAN, have been introduced. Rumi-GANs leverage negative examples to enhance the generator's ability to learn positive examples. We evaluate the performance of two controllable GAN variants, CGAN and Rumi-GAN, in generating game levels targeting specific constraints of interest: playability and controllability. This evaluation is conducted under two scenarios: with and without the inclusion of negative examples. The goal is to determine whether incorporating negative examples helps the GAN models avoid generating undesirable outputs. Our findings highlight the strengths and weaknesses of each method in enforcing the generation of specific conditions when generating outputs based on given positive and negative examples.

Introduction

In traditional generative models without conditioning, there is limited control over the attributes of the generated data. This limitation is effectively addressed by conditional models, where additional information can condition the data generation process towards specific outcomes. Recent research by Asokan and Seelamantula (2020) introduces a method where GAN models are trained using both positive and negative examples to guide and avoid specific outcomes (Rumi-GAN). Positive examples are data samples that align well with the desired outcome or target distribution. These examples represent the ideal or goal that the GAN is trying to achieve. Negative examples, on the other hand, are data samples that do not meet the desired outcome or diverge from the target distribution.

This study investigates the performance of two controllable GAN generators (CGAN and Rumi-GAN) through three sets of experiments conducted in the context of two 2D tile-based games. In each experiment, our goal is to enforce the generation of specific constraints within each game level. The first experiment aims to generate playable levels, emphasizing playability as the targeted constraint. The second experiment builds on the first by controlling a game-related feature, such as the specific number of pipes or treasures within the levels, while maintaining the playability constraint.

The first controllable model, a Conditional GAN (CGAN), is trained with levels containing both desired and undesired conditions (positive and negative examples), incorporating additional label information about the level conditions into the data. The second model, Rumi-GAN, also receives levels with both desired and undesired conditions and employs a loss function to encourage the generation of level segments with desired conditions while discouraging those with undesired conditions. Additionally, we utilize a baseline control model, a vanilla GAN, which cannot incorporate negative examples and, therefore, only receives levels with the desired condition for training.

Our results show that incorporating negative examples in training GAN models can help enforce some constraints of interest like playability. To our knowledge, this study is the first attempt to systematically compare various controllable GAN models and evaluate their effectiveness in conditioning generated outputs. Additionally, the integration of negative examples in GAN training (Rumi-GAN) has previously been only utilized on state-of-the-art computer vision datasets like MNIST, CelebA, and CIFAR-10, and this work is the first to apply this new approach to game-level segments.

The codebase of the project (including the training database, trained models, and generated artifacts) is available on GitHub[1]

Related Work

A range of studies have explored the use of GAN models for game level generation. Kumaran, Mott, and Lester (2020) developed a GAN-based architecture capable of generating levels for multiple distinct games from a common gameplay action sequence. Volz et al. (2018) trained a GAN to generate Super Mario Bros levels, using a variety of fitness functions to guide the search for levels with specific properties. Giacomello, Lanzi, and Loiacono (2018) applied GANs

[1] https://github.com/MahsaBazzaz/NegativeExamples

to learn a model of DOOM levels, finding that the inclusion of topological features improved the similarity of generated levels to human-designed ones. Capps and Schrum (2021) focused on generating whole Mega Man levels by using multiple GANs to model levels with different types of segments. Schrum et al. (2020) used interactive exploration of the GAN latent space to generate Mario levels and dungeons. Awiszus, Schubert, and Rosenhahn (2020) proposed TOAD-GAN that expands SinGAN architecture, enabling it to create token-based levels while being trained on a single example.

Conditional GANs have also been utilized in image generation and more recently in game level generation. Torrado et al. (2020) proposed a new GAN architecture, CESAGAN, for video game level generation, which incorporated an embedding feature vector input to condition the training of the discriminator and generator. This approach also introduced a bootstrapping mechanism to reduce the number of levels necessary for training and generate a larger number of playable levels with fewer duplicates. Kelvin and Anand (2020) applied conditional generative adversarial networks (CGANs) to create road maps, providing a balance between user control and automatic generation. Hald et al. (2020) furthered the application of GANs by using parameterized GANs to produce levels for a puzzle game, although they encountered challenges in approximating certain conditions.

Outside the domain of game level generation, Asokan and Seelamantula (2020) introduces a novel method for training GANs called "Rumi-GAN". This approach incorporates the concept of negative samples, which the GAN learns to avoid. The Rumi-GAN enhances the discriminator's capability to accurately model the target distribution and expedites the learning process of the generator. This method has been validated through various experiments, demonstrating its effectiveness, especially in scenarios involving imbalanced datasets where certain classes are under-represented.

Inspired by this approach, we aim to apply negative examples to the generation of game levels. Specifically, we intend to compare the performance of vanilla GAN, CGAN, and Rumi-GAN when provided with both positive and negative examples of game levels.

Domains

This work uses the Sturgeon (Cooper 2022b) constraint-based level generator to create a corpus of Super Mario Bros (Nintendo 1985) level segments. These segments are 14×32 in size and are based on the level 1-1 from the VGLC (Summerville et al. 2016). We have also created a corpus of a custom game called `Cave` which is a 14×14 simple top-down cave map and it was first introduced by Cooper (2022a) using tiles from Kenney (Kenney 2022).

In this work, we used two of Sturgeon's constraint-based features to generate our corpus. First is the ability to add constraints on the number of specific tiles. Second is (in addition to generating playable levels) the ability to create unplayable levels using an unreachability constraint (Cooper and Bazzaz 2024) that are similar to playable levels in local tile patterns, but are not possible to play.

Mario	
Type	Symbol
Ground	X
Breakable	S
Empty	-
Question Block	Q
Top Left Pipe	<
Top Right Pipe	>
Left Pipe	[
Right Pipe	]
Start	{
End	}

Cave	
Type	Symbol
Solid	X
Empty	-
Treasure	2
Start	{
End	}

Table 1: Different tile types and corresponding symbols present in each game.

We created 3000 each of playable and unplayable Mario segments with exactly 1,2, and 3 pipes in them, resulting in 9000 playable segments and 9000 unplayable segments of Mario. We also created 3000 each of playable and unplayable Cave segments with exactly 1, 2, and 3 treasures in them, resulting in also 9000 playable segments and 9000 unplayable segments of Cave.

Table 1 shows different tile types and corresponding symbols in each game. The "start" and "end" tiles are specifically kept in each level as the minimum requirement for level playability. We use one-hot encoding of these level segments during GAN training.

System Overview

All three controllable GANs adopt the Deep Convolutional GAN architecture as used by Volz et al. (2018), which is based on the original work of Arjovsky, Chintala, and Bottou (2017). This architecture employs batch normalization in both the generator and discriminator after each layer, ReLU activation functions for all layers in the generator (instead of Tanh), and LeakyReLU activation in all layers of the discriminator.

These models are trained using the WGAN algorithm. Wasserstein Generative Adversarial Networks offer an alternative to traditional GAN training, providing more stable training as demonstrated by Arjovsky, Chintala, and Bottou (2017). Both the generator and discriminator are trained with RMSprop, using a batch size of 32 and a default learning rate of 0.00005 for 200 iterations.

The following sections detail each controllable model, describing the classes of data used for training and the objective function of the training process.

Vanilla Generative Adversarial Nets

The vanilla GANs as introduced by (Goodfellow et al. 2014), consist of two models: a generative model (G) that captures the data distribution, and a discriminative model (D) that estimates the probability of a sample being real (from the training data) or fake (from the generative model). The entire training process is framed as a minimax game, where the discriminator tries to maximize the objective func-

tion, as described in Equation 1 (Goodfellow et al. 2014), while the generator tries to minimize it. Here, $x \sim p_d$ represents samples from the real data distribution (positive examples), and $x \sim p_g$ represents samples from the generator's distribution.

$$\min_{p_g} \max_{D(\boldsymbol{x})} \left(\mathbb{E}_{\boldsymbol{x} \sim p_d}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D(\boldsymbol{x}))] \right) \tag{1}$$

In this model, controllability is achieved by training a separate model for each set of desired constraints. For instance, with the goal of generating Mario levels with a specific number of pipes, a vanilla GAN is trained specifically to generate Mario level segments containing only one pipe. This means that the model is trained exclusively on level segments that feature only a single pipe.

Conditional Generative Adversarial Networks

CGANs (Mirza and Osindero 2014) apply extra information y to both the discriminator and generator. As shown in Equation 2 (Mirza and Osindero 2014) The loss function of a CGAN is an extension of the vanilla GAN loss function, incorporating this conditional information. This again results in in a minimax game in which the generator tries to generate realistic data conditioned on y, while the discriminator tries to distinguish between real and generated data, also conditioned on y. Here $x \sim p_{data}$ includes both positive and negative samples, distinguished by label y.

$$\begin{aligned} \min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x} \mid \boldsymbol{y})] \\ + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z} \mid \boldsymbol{y})))] \end{aligned} \tag{2}$$

Since positive and negative examples vary depending on the training target, a specific CGAN must be trained for each desired condition in each experiment. The only difference between these models is the label of the examples, which changes based on the model's objective.

Rumi Generative Adversarial Nets

Rumi-GAN (Asokan and Seelamantula 2020) is a specialized type of GAN inspired by the Sufi poet Rumi's philosophy of learning from both positive and negative experiences. Equation 3 (Asokan and Seelamantula 2020) shows how in this approach data distribution p_d is split into the target distribution that the GAN is required to learn (positive samples, p_d^+) and the distribution of samples that it must avoid (negative samples, p_d^-). The fake distribution which are the samples drawn from the generator p_g is there as before. α^+ and α^- is a weighting factor for the positive and negative real data distribution term. We set α^+ to 1 and α^- to 0.5.

$$\begin{aligned} \mathcal{L}_D^S = -\Big(\alpha^+ \mathbb{E}_{\boldsymbol{x} \sim p_d^+}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D(\boldsymbol{x}))] \\ + \alpha^- \mathbb{E}_{\boldsymbol{x} \sim p_d^-}[\log(1 - D(\boldsymbol{x}))] \Big) \end{aligned} \tag{3}$$

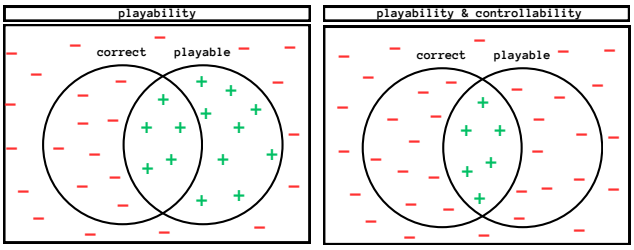

Figure 1: Venn diagrams of the sample space. Green plus signs represent positive samples and red minus signs represent negative samples provided to models.

Again, since positive and negative examples vary depending on the training target, we need to train a specific Rumi-GAN for each desired constraint.

Experiments

We conducted two different experiments to examine two sets of constraints we would like to see in the generated levels: playability and controllability. A playable level is a level such that there exists a path between the level's start and end locations. Controllability is derived from the correctness of the number of controlled features (eg. pipes or treasures) at each level. A level is considered correct if it has the desired number of features. Figure 1 shows the Venn diagram of the sample space. The sample space is divided into `playable-correct`, `playable-incorrect`, `unplayable-correct`, and `unplayable-incorrect` subspaces. Positive and negative samples are chosen from these subspaces according to the model's objective. In the experiments, negative examples are added in addition to the positive data that models utilize. To ensure an unbiased distribution of training data for models that incorporate negative examples, an equal number of positive and negative examples are sampled for training. Table 2 the models trained in each experiment and the exact input data of each model.

Experiment One: Playability

In this experiment, the goal is to ensure playability as the primary constraint for the models to meet. This means that we take the positive samples(p_d^+) from the `playable` subspace and negative samples (p_d^-) from the `unplayable` subspace. Both Conditional GAN and Rumi-GAN use this negative distribution as additional information.

Experiment Two: Playability and Controllability

This experiment's goal is to enforce both the number of some features in the level (number of pipes in Mario and number of treasures in Cave) and playability as a constraint for the models to satisfy. This approach gets positive samples (p_d^+) from the `playable-correct` subspace and negative samples(p_d^-) from the `playable-incorrect`, `unplayable-correct`, and `unplayable-incorrect` subspaces. This means Rumi-GAN, and Conditional GAN get the playable samples

	Model	Goal	Input Data
Experiment 1	Vanilla GAN	playable segments	playable segments of class 1, 2, and 3
	Rumi-GAN	playable segments	playable segments of class 1, 2, and 3 (+) unplayable segments of class 1, 2, and 3 (-)
	CGAN	playable segments	playable segments of class 1, 2, and 3 with labels (1,0), (2,0), (3,0) unplayable segments of class 1, 2, and 3 with labels (1,1), (2,1), (3,1)
Experiment 2	Vanilla GAN	generation of class 1	playable segments of class 1
		generation of class 2	playable segments of class 2
		generation of class 3	playable segments of class 3
	Rumi-GAN	generation of class 1	playable segments of class 1 (+), class 2 (-), class 3 (-) unplayable segments of class 1, 2, and 3 (-)
		generation of class 2	playable segments of class 1 (-), class 2 (+), class 3 (-) unplayable segments of class 1, 2, and 3 (-)
		generation of class 3	playable segments of class 1 (-), class 2 (-), class 3 (+) unplayable segments of class 1, 2, and 3 (-)
	CGAN	generation of class 1	playable segments of class 1 label (1,0) playable segments of class 2, and 3 with labels (2,1) and (3,1) unplayable segments of class 1, 2, and 3 with labels (1,1), (2,1), and (3,1)
		generation of class 2	playable segments of class 2 label (2,0) playable segments of class 1, and 3 with labels (1,1) and (3,1) unplayable segments of class 1, 2, and 3 with labels (1,1), (2,1), and (3,1)
		generation of class 3	playable segments of class 3 label (3,0) playable segments of class 1, and 2 with labels (1,1) and (2,1) unplayable segments of class 1, 2, and 3 with labels (1,1), (2,1), and (3,1)

Table 2: Input data and goals of trained models in experiment 1. The positive and negative signs indicate the positive and negative examples in Rumi-GAN.

with the desired condition as the positive examples to learn, and playable samples without the condition (other playable classes) and all unplayable classes as negative examples.

Evaluation

After the training models in each experiment, we generated 500 levels with each trained model and we evaluated each model based on the criteria of that experiment.

Experiment One: Playability

To evaluate Experiment One, we measure the percentage of playable levels as the metric. We use Sturgeon to find (if available) the shortest path between the start and goal of the level segments. Generated level segments that don't have a start or end, or have multiple starts and ends count as unplayable levels immediately. Table 3 shows the results of this experiment. In Mario both models using negative examples show better performance than vanilla GAN. In Cave, only Rumi-GAN takes advantage of the negative examples resulting in better performance compared to the other models.

	Vanilla GAN	Rumi-GAN	CGAN
Mario	67.8%	72%	**75.4%**
Cave	87%	**89.6%**	66.6%

Table 3: Experiment One Results. Percentage of playable results.

Experiment Two: Playability and Controllability

To evaluate Experiment Two, we measure the percentage of levels that have the correct number of pipes/treasures while being playable. This means unplayable levels with the correct number of pipes/treasures, or playable levels with an incorrect number of pipes/treasures count as failures in this experiment as they only achieved half of the objective. Note that for the purposes of evaluating the number of treasures in Cave, it is only the presence of the treasures that matters, not whether they are reachable. Tables 4 and 5 present the results of this experiment. Incorporating both the playability constraint and the number of pipes/treasures, as expected, makes generating correct outputs more challenging than in Experiment One. This increased difficulty leads to fewer `playable` levels being produced by each model compared to Experiment One. As a result, there are also fewer `playable-correct` outputs. Overall, while the inclusion of negative examples slightly improved performance in CGAN for Mario levels, it did not provide any benefit for Cave levels.

It's important to note that the results of Experiment Two

	correct				playable				playable correct			
	1	2	3	Avg	1	2	3	Avg	1	2	3	Avg
Vanilla	44.8	41.2	19.0	35.0	67.6	65.0	64.4	65.6	30.4	28.8	12.8	24.0
Rumi	41.2	41.0	0.6	27.6	65.2	71.2	56.8	64.4	25.4	31.0	0.0	18.8
Conditional	44.4	31.2	34.8	**36.6**	65.4	69.8	62.4	**65.8**	29.4	22.4	23.4	**25.0**

Table 4: Experiment Two Results. Detailed percentage of playable and correct outputs of Mario

	correct				playable				playable correct			
	1	2	3	Avg	1	2	3	Avg	1	2	3	Avg
Vanilla	24.4	20.6	4.6	16.5	85.0	79.0	81.8	81.9	20.6	16.2	4.0	**13.6**
Rumi	18.4	24.6	1.8	14.9	83.6	84.0	82.2	**83.2**	16.2	21.4	1.8	13.1
Conditional	38.6	30.0	19.6	**29.4**	41.7	31.9	32.2	35.3	16.0	9.6	12.8	12.8

Table 5: Experiment Two Results. Detailed percentage of playable and correct outputs of Cave

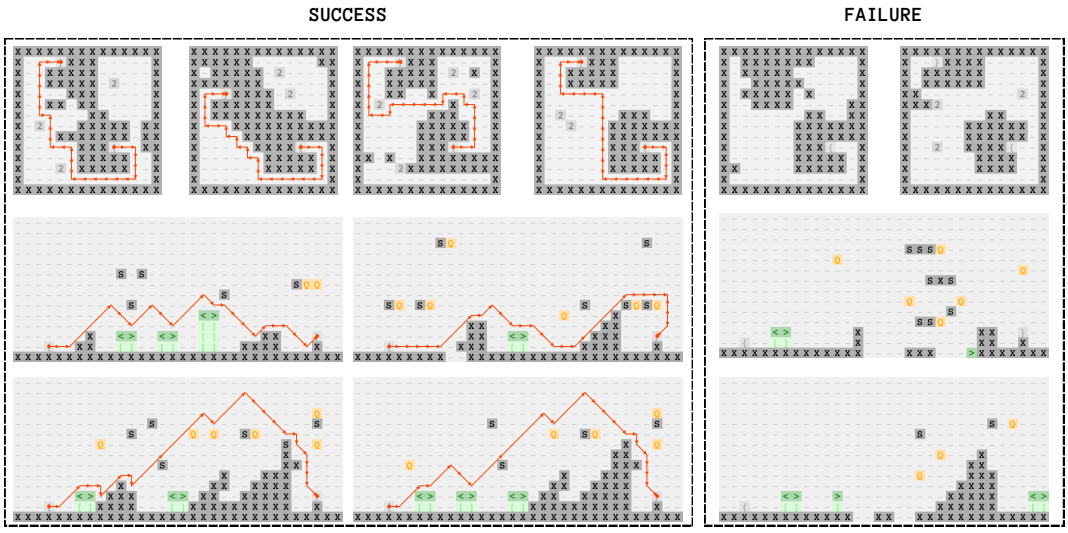

Figure 2: Examples of successful and failed levels generated in Experiment 1. A successful level is defined as a playable level. In contrast, unplayable levels either lack start or end positions or have blocked player paths.

reinforce the findings from Experiment One, with the same models—CGAN and RUMI-GAN—showing superior performance in enforcing playability constraints. Specifically, CGAN emerged as the leading model in terms of constraint correctness. However, the results did not indicate a clear winning model for enforcing both playability and correctness constraints. We believe this suggests that when combining different constraints, the model must be able to distinguish between the negative examples of each constraint. Otherwise, as shown in Experiment Two, the model may not derive significant benefits from the combination.

Discussion

The results of Experiment One demonstrate that incorporating negative examples in training GAN models can enhance their ability to generate more playable levels. In future work, we aim to reinforce this approach using levels annotated with players' paths. The surprising decrease in the performance of GAN models using negative examples in Experiment Two, may suggest the importance of the quality of the negative examples. In future work, we would like to explore multi-stage training approaches with high-quality negative examples in fine-tuning steps. We believe this approach could be effective when combined with bootstrapping methods (Torrado et al. 2020), or active learning methods with minimal training levels (Bazzaz and Cooper 2023). These approaches could make the training easier, with the only price for the additional controllability being the number of models trained on the minimal data.

Conclusion

This study explores the potential advantages of integrating negative examples into Generative Adversarial Networks (GANs) to enhance the generation of game levels. Inspired

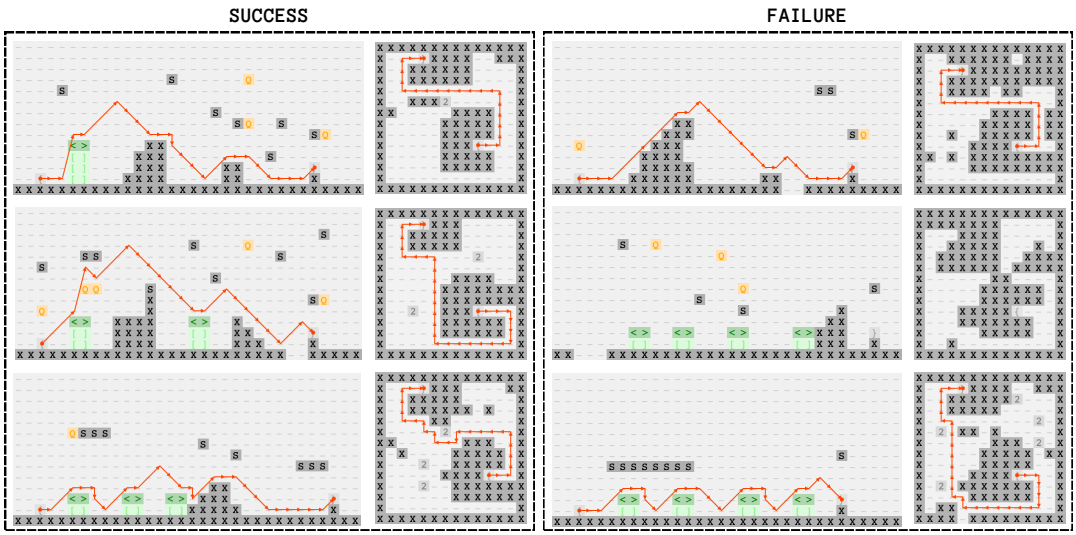

Figure 3: Examples of successful and failed levels generated in Experiment 2. A successful level must have the exact correct features (pipes/treasures) and be playable. Failed levels may be playable but with an incorrect number of features, or unplayable but with the correct number of features.

by the work of Asokan and Seelamantula (2020) on Rumi-GAN, the primary focus lies in leveraging positive examples to guide GANs toward producing desired outputs and negative examples to avoid undesirable outputs. Through comparative analyses involving Conditional GANs (CGANs), Rumi-GANs, and a baseline vanilla GAN with and without negative examples, it was observed that incorporating negative examples improves the capability of models to generate more playable outputs. However, this enhancement does not necessarily aid in enforcing constraints related to the controllability of specific features, such as the number of features in game levels.

Acknowledgments

Support provided by Research Computing at Northeastern University (https://rc.northeastern.edu/).

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein Generative Adversarial Networks. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 214–223. PMLR.

Asokan, S.; and Seelamantula, C. 2020. Teaching a gan what not to learn. *Advances in Neural Information Processing Systems*, 33: 3964–3975.

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

Bazzaz, M.; and Cooper, S. 2023. Active learning for classifying 2d grid-based level completability. In *2023 IEEE Conference on Games (CoG)*, 1–4. IEEE.

Capps, B.; and Schrum, J. 2021. Using multiple generative adversarial networks to build better-connected levels for mega man. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 66–74.

Cooper, S. 2022a. Constraint-based 2d tile game blending in the sturgeon system. In *Proceedings of the Experimental AI in Games Workshop*.

Cooper, S. 2022b. Sturgeon: tile-based procedural level generation via learned and designed constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, 26–36.

Cooper, S.; and Bazzaz, M. 2024. Literally Unplayable: On Constraint-Based Generation of Uncompletable Levels. In *Proceedings of the 19th International Conference on the Foundations of Digital Games*, 1–8.

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.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

Hald, A.; Hansen, J. S.; Kristensen, J.; and Burelli, P. 2020. Procedural content generation of puzzle games using conditional generative adversarial networks. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 1–9.

Kelvin, L. Z.; and Anand, B. 2020. Procedural Generation of Roads with Conditional Generative Adversarial Networks.

In *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, 277–281.

Kenney. 2022. Free game assets. https://www.kenney.nl/assets. Accessed: 2022-01-07.

Kumaran, V.; Mott, B.; and Lester, J. 2020. Generating Game Levels for Multiple Distinct Games with a Common Latent Space. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15: 102–108.

Mirza, M.; and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Nintendo. 1985. Super Mario Bros. Game [NES].

Schrum, J.; Gutierrez, J.; Volz, V.; Liu, J.; Lucas, S.; and Risi, S. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 148–156.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The Video Game Level Corpus. *arXiv:1606.07487 [cs]*.

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)*, 41–48. IEEE.

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.