Maastricht University

Master Thesis

Utilizing generative adversarial networks for level generation in a physics-based simulation.

by

Frederic Marvin Abraham
i6262598

DKE: Dr. Matthew Stephenson

University of Maastricht
Data Science & Knowledge Engineering
Maastricht, September 20, 2022

Contents

Abbreviations

ML Machine Learning

NN Neural Network

CNN Convolutional Neural Network

DL Deep Learning

RMSP Root Mean Squared Propagation

EM Earthmover Distance

MLP Multilayer Perceptron

LSTM Long Short-Term Memory

RNN recurrent neural network

VAE Variational Autoencoder

GAN Generative Adversarial Network

CGAN Conditional GAN

WGAN Wasserstein GAN

WGAN-GP Wasserstein GAN with gradient penalty

DCGAN deep convolutional generative adversarial network

MC mode collapse

PCG Procedural Content Generation

PCGML Procedural Content Generation via machine learning

MNIST Modified National Institute of Standards and Technology database

VGLC Video Game Level Corpus

LVE latent variable evolution

CMA-ES Covariance Matrix Adaptation Evolutionary Strategy

Abstract

1 Introduction

The field of procedural content generation (PCG) has a long history in video game development, initially used to compress game data Amato 2017. PCG describes the creation of content automatically through algorithmic means Yannakakis and Togelius 2011.

Machine-learning-based PCG uses models trained from existing video game content and has received increasing research attention recently Liu et al. 2020.

Different types of networks have been used to generate content. Previously, generative adversarial networks (GAN) were successfully used in tile-based games such as Super Mario with no physical constraints Volz et al. 2018. The domain this thesis will focus on is a 2D physics-based real-valued block simulation in the form of levels in science birds.

GANs describe the construct of two adversarial networks, which play a min-max game of one network that generates new content and a second that discriminates the generated content to differentiate between real and generated Goodfellow et al. 2014. The generated content can use almost any underlying data structure, and therefore the two networks can be a variety of different architectures depending on the data structure. The variety allows the testing of different level representations suitable for the science bird domain and, therefore, different neural network architectures need to be implemented.

Further, a genetic algorithm can be used to evolve the latent variable Tanabe et al. 2021. By defining different testing metrics such as stable, beatable by AI, horizontal and vertical constraints or the frequency of different blocks, the algorithm can explore the latent space for levels with these different characteristics.

Write something about AI birds competiton

1.1 Motivation

A significant portion of the budget in video games is spend to create media used in every aspect of a game. The development of better generators able to aid a content creator will improve their output and consequently reduce the cost of video game development Amato 2017.

The idea of a GAN, two adversarial networks training each other, can be adapted to new domains, by changing the kind of data and networks used. The range and variety of different applications of GANs, such as images, terrain, faces and video game levels motivates its research. The domain mentioned above with the physical constraint differentiates itself from the usual application environment of GANs. Therefore, exploring new domains reduces the transfer work required in applications relying on a similar data structure or having similar external constraints. For example the 2D physics-based real-valued block simulation could be extendible into the third dimension or more complex shapes.

Game-playing agents could also benefit from having a good content generator. A GAN that can produce solvable and challenging levels could help train the AI which benefits from a variety of levels. At the same time, a improved AI could enhance the level generator with better validation capabilities.

1.2 Research Goals

This description of the task concludes in the research goals of exploring different encodings for the science bird level description that can be used in a generative adversarial network and consequently the implementation of different GAN architectures. Further, the exploration of the latent space and evaluation of the generated levels through different metrics is required to access the performance of the different architecture and representations.

1.3 Problem Statement

Angry Birds is often targeted in PCG as a challenging game domain for level generation (https://aibirds.org/level-generation-competition.html)

1.4 Research Questions

2 Related Work

Procedural Content Generation (PCG) refers to the creation of content automatically through algorithmic means (Yannakakis and Togelius 2011). Over the years, the field of PCG became populated with various algorithms aiming to archive different goals. The original problem PCG addressed was memory limitations for storing larger video game levels on computers in the 1980s (Fontaine et al. 2020). One of the earliest games that incorporated PCG to generate adventures and levels was "Beneath Apple Manor" in 1978 (Doull 2015) followed by the more famous and highly influential game Rogue (Doull 2016) in 1980, which is the progenitor of following games in the "Rouguelike" category. The original dungeon generation algorithm created a level by generating several rooms connected with corridors in a 2d grid-based environment. When new game development focused on more realistic graphics in the 1990s, procedural modelling algorithms concentrated on generating, e.g. textures, plants or terrain (Smelik et al. 2014).

The master thesis focuses on the more recent area of Procedural Content Generation via machine learning (PCGML) which utilises the latest advancements in Machine Learning (ML) to generate new content. PCGML is defined as the generation of game content using machine learning models trained on existing content (A. Summerville et al. 2017).

The application of ML, in particular Deep Learning (DL), led to increased capabilities and application methods to learn from a large amount of data and has won numerous contests in pattern recognition. (Schmidhuber 2015) The major difficulty in many of the problems DL excels in is a vast amount of features with different levels of importance. Goodfellow et al. 2016 describe DL as the solution to the central problem in representation learning by introducing representations expressed in terms of other, simpler representations.

The Figure 2.1 created by Goodfellow et al. 2016 based on the work of Zeiler and Fergus 2013 on visualizing and understanding Convolutional Networks describes exemplary how a deep learning model can learn the concept of a person. The abstract visualized Multilayer Perceptron (MLP) or feedforward deep Neural Network (NN) takes the sensory input such as an image and learns different representations on

different layers. The complex mapping of an input image to an object is split into multiple smaller tasks.

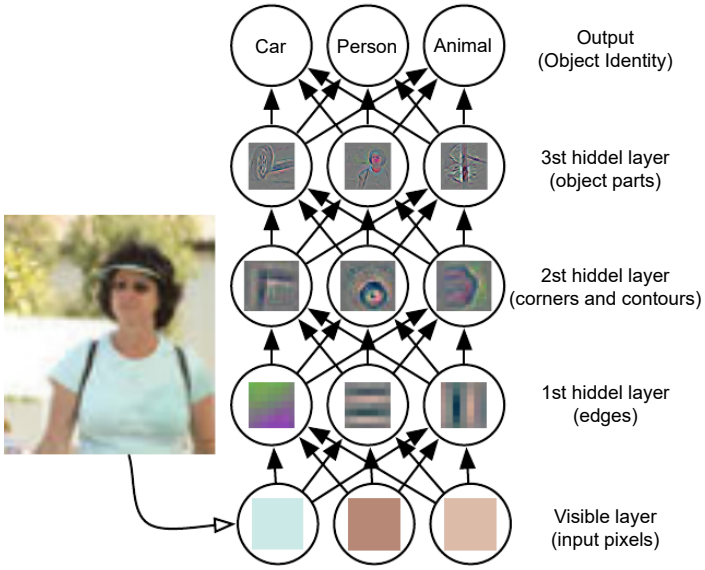

Figure 2.1: Example of a deep learning classification (Zeiler and Fergus 2013)

After training the neural network on an image classification task, the first layer learned to detect edges. The second layer combines the representation of edges into corners and contours, while the last layer detects the presence of various object parts. The output layer takes the existence of different object parts and uses them to recognise different objects.

In PCGML, various models are used for content generation. A. Summerville et al. 2017 surveyed machine learning methods for content generation such as long short-term memory (LSTM) networks, autoencoders, deep learning and generative adversarial networks. The other category of content generation methods falls into the search-based methods, for example, evolutionary algorithms, solver-based methods, which try to maximize a objective while preserving a specific constraint, and constructive methods using grammars. Other machine learning models used in PCG that are not based on neural networks are Markov-Models, one-dimensional n-gram models, clustering, and matrix factorization (A. Summerville et al. 2017).

The main focus of this master thesis is to utilize generative adversarial networks to generate levels in a real valued block world. Therefore the following section explain the main concepts of GANs, their advancements in architecture and the variety of training methods.

2.1 Generative Adversarial Networks

The chief AI Scientist at Facebook LeCun 2016 described Generative Adversarial Network (GAN) as “the most interesting idea in the last ten years in Machine Learning”. The main adversarial idea, which is the basis of GANs, has been successfully applied in many areas, such as machine learning, artificial intelligence, computer vision and natural language processing. (Gui et al. 2021) From a game-theoretical point of view is the adversarial idea fundamentally a game between two opposing systems trained in an adversarial manner to reach a zero-sum Nash equilibrium (Moghadam et al. 2021). Another public example of how an adversarial system exceeded previous achievements is when the AlphaGo model (Silver et al. 2016) defeated the top human Go player. Parts of AplhaGo utilized two networks that were trained by playing against itself. Figure 2.2 shows progress in GANs capabilities from the year 2014 to 2018.

Figure 2.2: Different GAN architectures in the task of face synthesis.
From left to right: (1) The Original GAN Paper (Goodfellow et al. 2014). (2) First use of Deep Convolution Networks (Radford et al. 2015). (3) Using a joint distribution training task (Liu and Tuzel 2016). (4) Progressive training of Generator and Discriminator (Karras et al. 2017). (5) Incorporates style transfer architecture in generator (Karras et al. 2018).

Goodfellow et al. 2014 introduced generative adversarial networks as a framework for estimating generative models via an adversarial process in which two models are trained simultaneously. He described GANs as frameworks consisting of two models, which can be any kind of network, instead of one coherent model architecture. This change is done in recent literature and can be referred to interchangeably. Figure 2.3 visualizes the general structure of a GAN framework consisting of the two models.

The two models mentioned above are the generator that tries to capture the data distribution of the training data and the discriminative model, also called the critic,

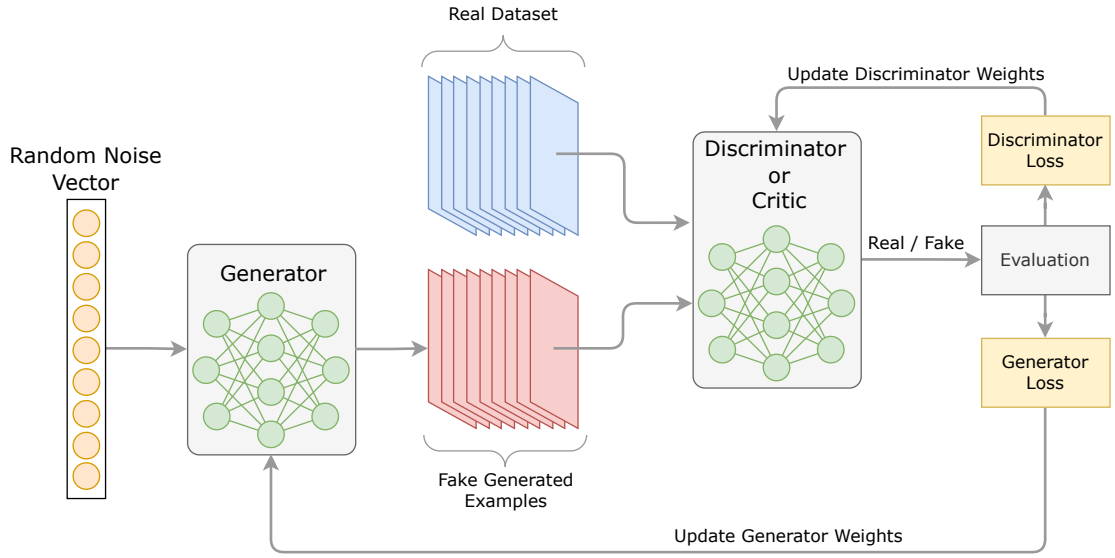

Figure 2.3: Overview of a generative adversarial network

which tries to differentiate between samples drawn from the training data and samples generated by the generator. The generator is trained to maximize the probability that the discriminator mistakes its generated example as drawn from the actual distribution. The GAN can be trained through backpropagation if the generator and the discriminator are MLP.

Goodfellow et al. 2014 describe the training process, visualized in figure 2.3, as a two-player minimax game in the following steps:

1. Create a noise variable $p_z(\mathcal{Z})$ which functions as input to the generator.
2. The generator, which tries to learn the distribution p_g over the data x, is defined as a mapping from the random input space into the data space $G(\mathcal{Z}; \theta_g)$. G is a differentiable function due to being an MLP with parameters θ_g.
3. The discriminator is defined as a function $D(x; \theta_d)$ that maps from the data space to a single scaler. The scaler $D(x)$ is defined as the probability of x being drawn from the real data distribution.
4. The discriminator is trained to classify real and fake data to maximize the probability of assigning the correct label. The error in assigned labels is the loss used to train the discriminator.
5. The generator tries to minimize $\log(1 - D(G(z))$. In other words, given the noise variable z, the generated example shall receive a probability close to 1 from the real dataset.

The description of the training process accumulates into the aforementioned two-player minimax game with the value function $V(G, D)$:

$$\min_G \max_D V(D, g) = \mathbb{E}_{x \in p_{data}(x)}[logD(x)] + \mathbb{E}_{z \in p_z(z)}[\log 1 - D(G(z))] \tag{2.1}$$

Solving this problem and "finding the Nash equilibrium is a very difficult problem. [As the] [...] cost functions are non-convex, the parameters are continues, and the parameter space is extremely high-dimensional." (Salimans et al. 2016)

2.1.1 Difficulties in GAN Training

Training a GAN has been repeatedly stated to be a challenging problem. (Arjovsky and Bottou 2017; Salimans et al. 2016; Gui et al. 2021) The main challenges that can be observed when training a GAN are mode collapse (Goodfellow et al. 2014), the discriminator loss converging quickly to zero and therefore does not provide a sufficient update to the generator (Arjovsky and Bottou 2017) and difficulties in making the generator and discriminator converge. (Radford et al. 2015)

2.1.1.1 Mode Collapse

The primary failure mode of GANs is the mode collapse (MC) problem, in which the generator collapses to a parameter setting in which it learns a mapping of different input z values to the same output point. (Salimans et al. 2016; Goodfellow 2017) Jabbar et al. 2020 describe it as the most crucial topic of GAN training. MC comes in different severity and is therefore classified into partial or complete mode collapse.

A partial mode collapse, visualized in figure 2.4, is the case where some diversity remains in the generator. In contrast, the complete mode collapse only produces only one data point. It can be observed that when a mode collapse is about to happen that all gradients of the generator point in similar directions. (Salimans et al. 2016) In the original GAN architecture, no mechanism enforces diversity, and it exists no coordination between the gradients of each example of the discriminator. Thus the discriminator calculates the gradients for each example independently and points the generator to the point that it believes to be the most probable. After the generator collapses and produces only a single or few distinct outputs, the gradient

1. The MNIST database of handwritten digits

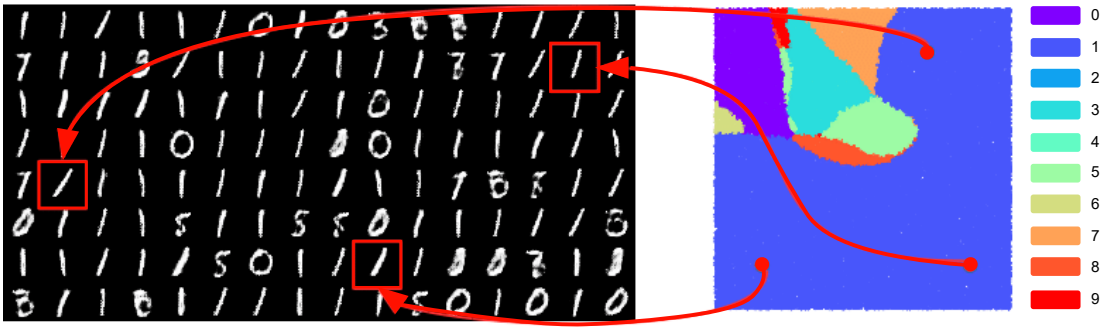

Figure 2.4: A partial mode collapse of a GAN trained on the MNIST[1]dataset, which is a hand drawing dataset of numbers. Visualized on the right is a two-dimensional projection of the input latent space $\mathcal{Z}$, which shows that a majority of the latent space vectors are mapped to the drawing of a one. The graphic is created by Tran et al. 2018.

descent algorithm used to train the models can not separate the outputs resulting in a GAN that can not converge to a correct distribution.

The solution to the mode collapse problem has many different approaches. The first simple solution to associate examples with one another is to introduce batch normalization (Ioffe and Szegedy 2015) as firstly done in the deep convolutional generative adversarial network (DCGAN) (Radford et al. 2015), which is further discussed in section 2.1.2.1. Batch normalization modifies the input for each layer to have zero mean and und unit variance over the whole batch. Radford et al. 2015 describe batch normalization as critical to get deep generators to begin to learn while preventing mode collapse. This approach is criticized by Gulrajani et al. 2017 to change the problem of the discriminator training. Instead of learning a mapping from a single input to a single output, it is changed to a mapping from an entire batch of inputs to a batch of outputs, reducing the quality of the outputs. They recommend layer normalization (Ba et al. 2016) as a drop-in replacement for batch normalization.

Another approach is to modify the loss function to encourage the generator to be more diverse. This is done in various ways, with the most adopted one being the Wasserstein Gan discussed in 2.1.2.2. A more direct approach by Tran et al. 2018 is to introduce a latent-data distance constraint which tries to enforce compatibility between the latent sample distances and the corresponding data sample distances.

2.1.1.2 Vanishing Gradients

The vanishing gradient problem is a general deep learning problem of NN (Basodi et al. 2020). It results in the problem that the generator does not improve in producing good quality images.

The main issue is that the gradients required to train the generator become vanishingly small in the initial layers of the network. Combined with the problem that minimizing $\log(1 - D(G(z))$ results in the issue that if the discriminator is too confident in its prediction, the probability becomes $D(G(\mathcal{Z})) \approx 0$ and the gradient diverges to zero. Goodfellow et al. 2014 proposed to maximize $D(G(\mathcal{Z}))$ instead, which provides stronger gradients in the early stages of the learning process but introduces a larger variance of gradience, making the training less stable.

Another thing done in practice, even in the original training algorithm, is training the discriminator more than the generator. The goal is to keep the discriminator close to its optimum for the intermediate generator state. Arjovsky and Bottou 2017 argue that the gradients become more reliable with a more trained discriminator. This becomes even more important in architectures that use the Wasserstein distance due to its stronger gradients, which will be discussed in section 2.1.2.2.

2.1.2 Development of GANs

The described difficulties in training GANs and their promising capabilities in generating a variety of content led to a variety of research into improving the stability in training, measurability and overall results.

Saxena and Cao 2020 categorize the different approaches for improving a GAN in three distinct ways.

1. Re-engineering the overall network architectures.
2. Proposing a new objective function for generator and or discriminator.
3. Developing new optimization algorithms for the generator and or discriminator.

Plenty of other optimizations which are non-GAN specific, such as Model Ensamble of multiple generators (Tao et al. 2018) or training multiple discriminators (Wang et al. 2016)

2.1.2.1 DCGAN

The aforementioned deep convolutional generative adversarial network (DCGAN) fall into the first category of reengineering the architecture of the generator and discriminator. It is the first Convolutional Neural Network (CNN) based GAN architecture that shows a continuous training process and performs well in image generation tasks. (Jabbar et al. 2020)

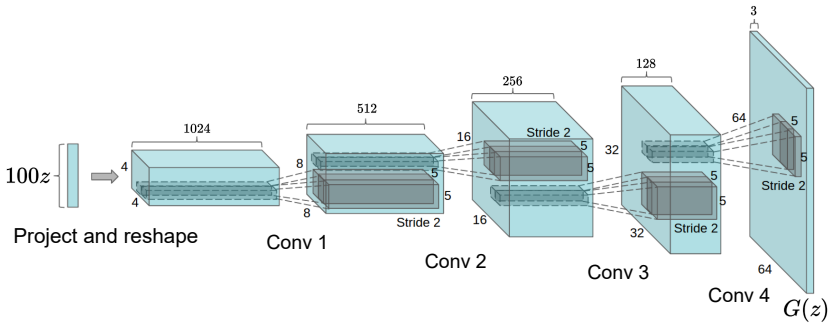

Figure 2.5: The architecture of a DCGAN generator that creates a 64x64 image. Visualization of the network is created by Radford et al. 2015

Figure 2.5 visualizes the architecture of the generator developed by Radford et al. 2015. Compared to the original GAN architecture that only used fully connected layers and pooling layers the proposed architecture uses transposed convolutions to create the required size. No fully connected or pooling layers are used. The use of fractionally-strided convolutional layers also called transposed convolution, improved the stability of GAN training significantly. (Jabbar et al. 2020) Similarly, the discriminator uses only convolutional layers to derive its prediction from a given image.

The generator uses the ReLU (Nair and Hinton 2010) activation function between the layers and a Tanh function at the output layer, similarly to the original implementation by Goodfellow et al. 2014. In contrast, the discriminator does not use the Maxout function but a leaky ReLu (Xu et al. 2015).

2.1.2.2 Wasserstein GAN

The Wasserstein GAN (WGAN) developed by Arjovsky et al. 2017 is an improvement of the training of GANs that falls into the second category of proposing a new objective function. The objective function they adapt for training

the Wasserstein GAN is the Earthmover Distance (EM) [2] or Wasserstein-1 distance.

The original discriminator objective function determines whether an example is drawn from the fake or real distribution and designates a value between 0 and 1, respectively. As Goodfellow et al. 2014 pointed out, this leads to vanishing gradients if the two distributions do not overlap, resulting in slow generator training. The EM distance is the distance between two probability distributions over a region D. It is intuitively described by visualizing both distributions as different ways to pile an amount of earth with the task of transforming one pile into the other. The EM describes the minimal cost of transformation, where the cost is the amount of earth times the distance by which it moved.

Compare to Wikipedia to discuss if cool

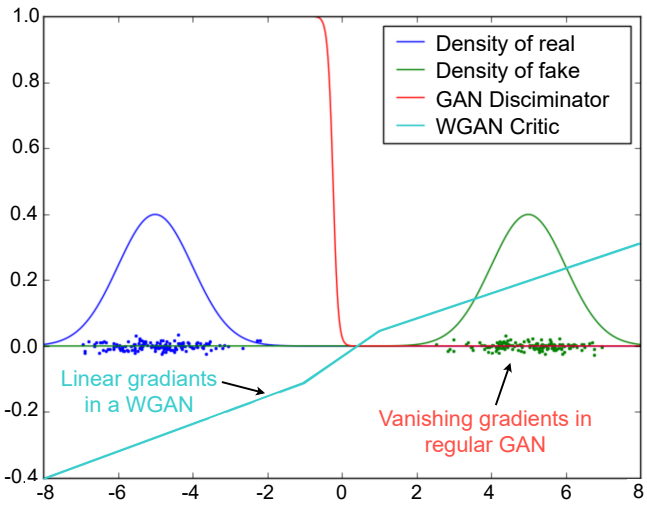

Figure 2.6: A artificial example in which two optimal trained discriminators/critics, trained to differentiate two Gaussians, calculate gradients.

Figure 2.6 gives insight into how the gradients behave for a WGAN-Critic using the Wasserstein-Distance / earth mover distance and the original GAN discriminator. The GAN discriminator's gradients saturate and result in vanishing gradients, while the WGAN critic provides clean gradients on all parts of the space (Arjovsky et al. 2017).
Incorporating the Wasserstein Distance into the gan training results in the gradients shown in table 2.1. When applying the Wasserstein Distance, the WGAN critic f becomes a function that has to be a 1-Lipschitz function, which is a strong form of uniform continuity. In other words, a Lipschitz continuous function is limited in how fast it can change, which results in good gradients even far away from the actual distribution.

2. Formal definition: The Earth Mover's Distance

	Discriminator/Critic	Generator
GAN	$\nabla_{\Theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(D(x^{(i)})) + log(1 - D(G(z^{(i)})))]$	$\nabla_{\Theta_g} \frac{1}{m} \sum_{i=1}^{m} log(D(G(z^{(i)})))$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} [f(x^{(i)})) - f(G(z^{(i)}))]$	$\nabla_\Theta \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)})))$

Table 2.1: Comparing the gradients for discriminator/critic and generator for the original GAN and Wasserstein Gan (Hui 2018).

The constraint is enforced by clipping the weights of the critic f by an extra hyperparameter c. The WGAN algorithm describes the weight update of the critic by using the Root Mean Squared Propagation (RMSP) (Hinton et al., n.d.) update and weight clipping:

$$w \leftarrow w + \alpha \cdot RMSProp(w, g_w)$$
$$w \leftarrow clip(w, -c, c)$$

Arjovsky et al. 2017 show multiple benefits over the original GAN training when using the Wasserstein optimisation function.

- Does not require a careful balance between the discriminator and generator.
- Less restricted in the architecture design of the NN.
- Mode collapse becomes less likely because of more stable gradients.
- The EM Distance correlates well with the sample quality.

While noting the progress that WGAN makes toward stable training of GANs, Gulrajani et al. 2017 criticise the use of weight clipping to enforce the Lipschitz constraint on the critic. They claim this could lead to low-quality samples or that the training is less likely to converge. They propose an alternative to weight clipping by penalising the norm of the critic's gradient with respect to its input. They enforce the 1-Lipschitz constraint by penalising the gradient by adding the penalty given in equation 2.2.

$$gp = \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{2.2}$$

The variable $\hat{x}$ is an interpolation between a generated sample and a sample drawn from the real distribution. Given that interpolation, the norm of the gradient of the critic at the interpolated point should be 1. They prove that an optimal critic contains interpolations with a gradient of 1.

They claim that the proposed Wasserstein GAN with gradient penalty (WGAN-GP) performs better than the standard WGAN with higher quality samples and more stable training through a wider variety of architectures with less hyperparameter tuning. A downside to this approach is that calculating the interpolations for each training step is an expensive operation.

2.2 Videogame level Generation via machine learning

With the basis of machine learning and GANs covered in section 2.1, this section reviews the application in the context of Procedural Content Generation via machine learning (PCGML), primarily video game level generation. PCGML consists mainly out of the two fields of data representation and training method. A. Summerville et al. 2017 organize PCGML in their taxonomy of methods techniques into these two categories.

The underlying data representation used in training describes how the data is encoded to be used for training and generation. In their taxonomy, they consider three distinct ways of data representation which can be used to represent data: (1) Sequences, (2) Grids, (3) and Graphs. One piece of information is not constrained to only one representation but can be encoded in many different ways. The Video Game Level Corpus (VGLC) (A. J. Summerville et al. 2016) contains 428 levels from 12 games in all three representations. Particularly the level of a platformer has been defined in all three representations (Summerville and Mateas 2016; A. Summerville et al. 2017).

The methods reviewed in their taxonomy are Back Propagation, Evolution, Frequency Counting, Expectation Maximization, and Matrix Factorization. The main focus of this section is level generation through GANs, which primarily use NN in their underlying architecture and are mainly trained through backpropagation.

2.2.1 General Level generation with GANs

Transferring the knowledge of image synthesis into level generation is not a trivial task. Primarily finding an encoding of the level that can be decoded even if the outcome of the GAN is noisy or imperfect is difficult.

One of the first to apply GANs in the context of level generation is Giacomello et al. 2018 in their work to generate DOOM levels based on human-designed content.

They extract six images for each level representing the game level file to train the GAN. Figure 2.7 visualizes the first four layers from left to right.

- A floor map represents the space a player can walk on or not walk on.
- A height map that defines value specifies the vertical height location of the floor.

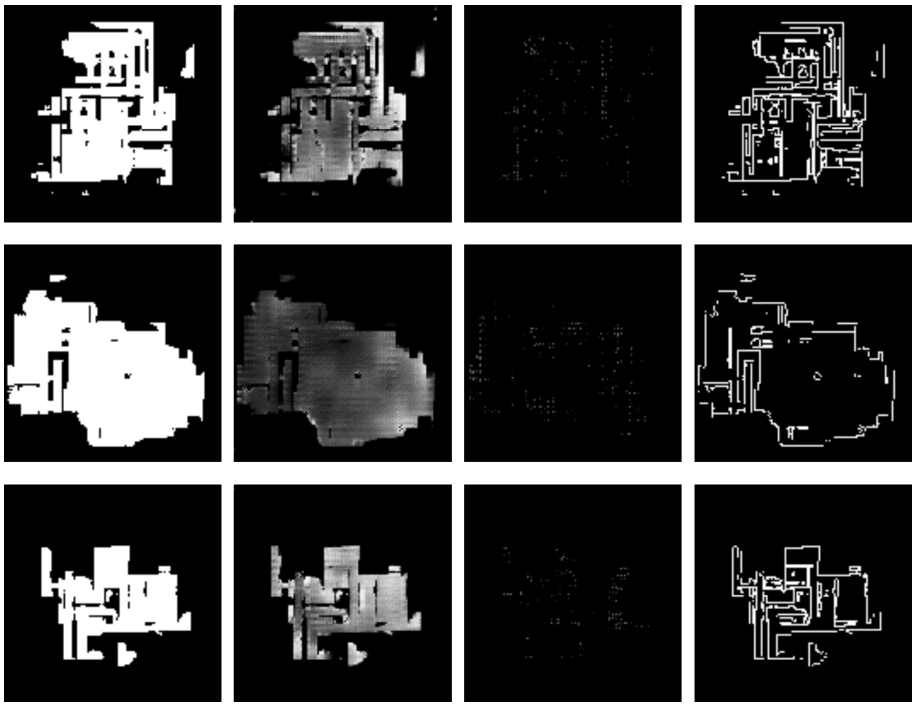

Figure 2.7: The layers required for recreating a doom level. From left to right: the FloorMap, HeightMap, ThingsMap, and WallMap of the generated levels.

- The things map includes one-pixel representations of locations where items are placed. Different items are represented through different values.
- A wall map visualizes the locations of walls through one-pixel borders.
- The trigger map encodes the location of triggers, e.g. doors or elevators.
- Lastly, a RoomMap segments a level into different rooms.

They describe their preliminary results as a good starting point for researching the viability of GANs compared to classical PCG. The generated levels, also the playability was not tested, contained DOOM typical features and are supposedly interesting to explore.

Volz et al. 2018 utilized GANs to generate complete Mario levels. They use Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) to search the latent space of the GANs generator to influence the outcome based on different metrics over the generated levels. The first approach is to optimize different block distributions. For example, fewer stone blocks could lead to an air level with greater difficulty. In their second approach, they utilize a Mario AI (Togelius et al. 2013) that can produce playthrough data of their generated levels. They focused on optimizing toward playable levels with a scalable difficulty. The idea of using latent

variable evolution (LVE) to explore the generator's latent space was firstly introduced by Bontrager et al. 2017 in their works to match generated fingerprints to as many real fingerprints as possible. Evolving the latent space to gain control over the output stands in contrast to Conditional GANs (CGANs) (Mirza and Osindero 2014), which utilize a condition vector combined with the noise vector as input to the generator to produce a controllable output.

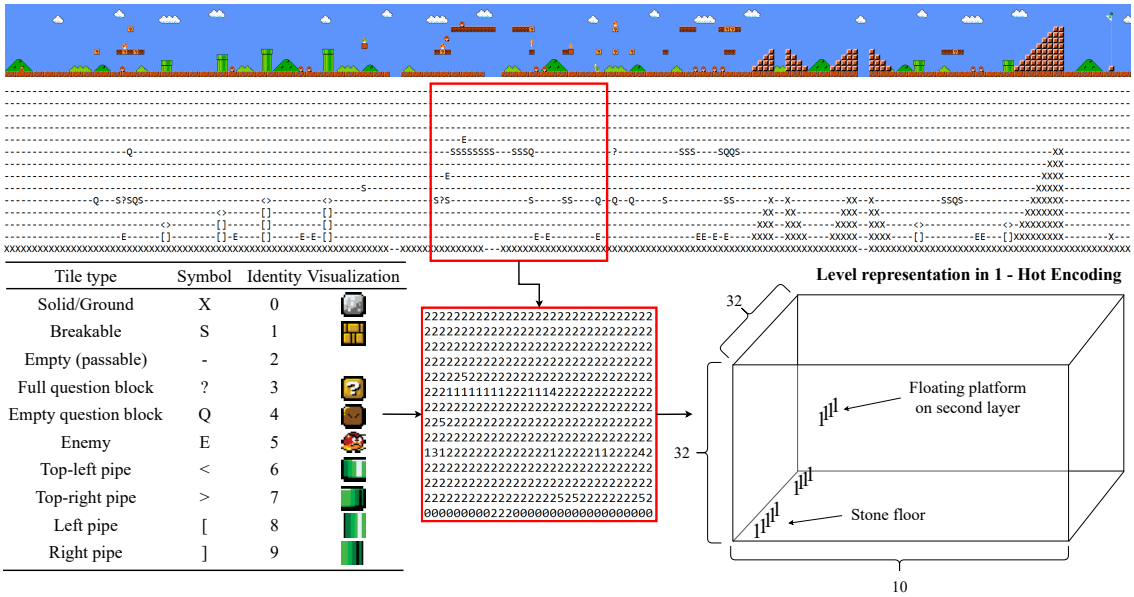

Figure 2.8: Process of transforming a Mario level into a One-Hot, multi-dimensional level representation used by MarioGAN. The $32 \times 32 \times 10$ matrix, on the bottom right, is filled with zeros except x, y coordinate on the layers of the blocks visible in the selected window.

Figure 2.8 visualizes how a Mario level is transformed into a representation that can be put into the discriminator. The ASCII representation provided by the VGLC (A. J. Summerville et al. 2016) is mapped into the identity matrix, which is put into a one-hot encoded matrix. In other words, each x, y coordinate becomes a vector of size 10 with a one at the location of the visible block. The advantage of this number-based representation is that the generated image can be decoded into a playable level through an argmax operation over the layers of the one hot encoded matrix.

They conclude that GANs can capture high-level structures of the training level even though they sometimes produce broken elements such as pipes and structures. Their main conclusion is that LVE is a promising approach for a fast level generation that can be adapted to other game genres.

2.2.2 Angry Birds Level Generation

The physics-based simulation, which is the subject of this thesis, is represented in the domain of physics-based puzzle games similar to the popular game Angry Birds. Procedural level generation research usually uses the Science Birds game[3] (Ferreira and Toledo 2014), a clone of Angry birds developed in Unity that provides an interface for remote AI execution. The procedural level generation field in this game domain is active, and a level generation competition is held yearly (Stephenson et al. 2019).

Maybe move to intoduction

Machine learning based level generation using neural networks and backpropagation for Angry Birds has only recently been tried by Tanabe et al. 2021. Conventionally level generation approaches for Angry Birds are designed using domain knowledge and search-based approaches (Tanabe et al. 2021).

Ferreira and Toledo 2014 used a genetic algorithm to find a stable combination of blocks and predefined structures.

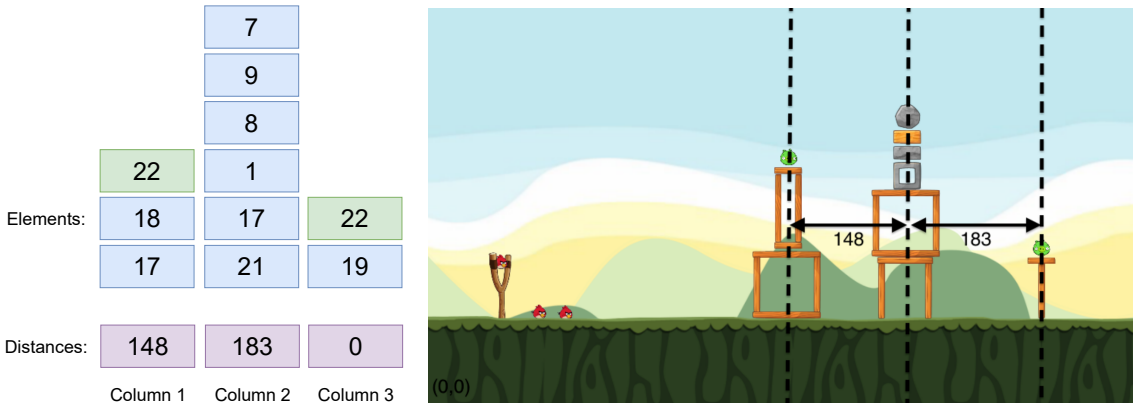

Figure 2.9: Level encoding defiend by Ferreira and Toledo 2014 used block and predefiend structures in arrays of columns. For example the block ID 22 represents pig.

Figure 2.9 shows how a level with three structures is represented as an individual used in the genetic algorithm. The initial population is generated by using a stochastic selection that defines the likelihood of a block appearing at the bottom, middle or top. Crossover operators are used only between whole columns, and mutation operators change individual blocks.

Winning entry for the 2017 and 2018 AIBIRDS level generation competitions[4] is the search-based approach by Stephenson and Renz 2017, which can create complex

3. Science Birds Source Code: https://github.com/lucasnfe/science-birds
4. https://aibirds.org/other-events/level-generation-competition.html

stable structures with various different elements. The proposed level generator is built upon and improves their previous iterations. (Stephenson and Renz 2016a, 2016b)

Structures generated using the original algorithm are made up of rows, each consisting of a single block type. It operates by recursively adding rows of blocks to the bottom of the previous row. After the block type of each row is selected, the previous row is divided into subsets based on the distance of the blocks in the row. For each subset, the block placement of putting the subsequent blocks in the centre, at the edge or in both locations are created and checked for validity (Overlap and support is checked). Of all possible valid block placements, one is selected at random.

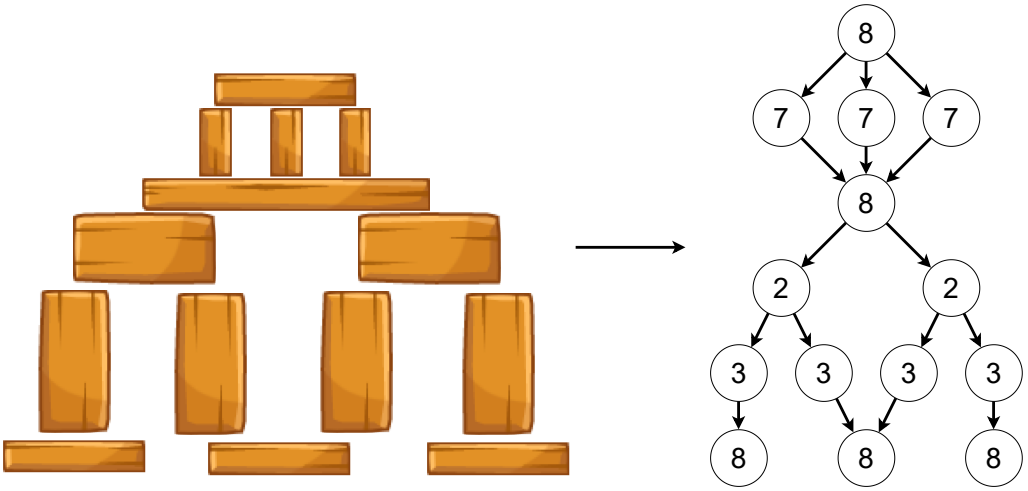

Figure 2.10: A structure generated by Stephenson and Renz 2016a which shows the block placements and how the structure can be represented as acyclic graph.

Figure 2.10 shows the three different kinds of block placements in different subsets incorporated into a generated structure. The pig placement is done by analysing the structure and searching for free space above the centre or corners of each block. This algorithm is extended in (Stephenson and Renz 2016b) to include irregular blocks after the pigs have been placed.

This approach's main limitation is to limit the block type of each row, which reduces the variety of possible structures. The last iteration (Stephenson and Renz 2017) extends the generator to allow multiple block types in one row by swapping blocks with other block types of the same height.

The aforementioned NN-based approach by Tanabe et al. 2021. uses Variational Autoencoder (VAE) (Kingma and Welling 2013) in combination with Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997). VAEs are trained on

Maybe talk about structural weak points and their shielding?

recreating the input at the output, and LSTMs are a version of recurrent neural networks (RNNs) and are usually used in text processing tasks.

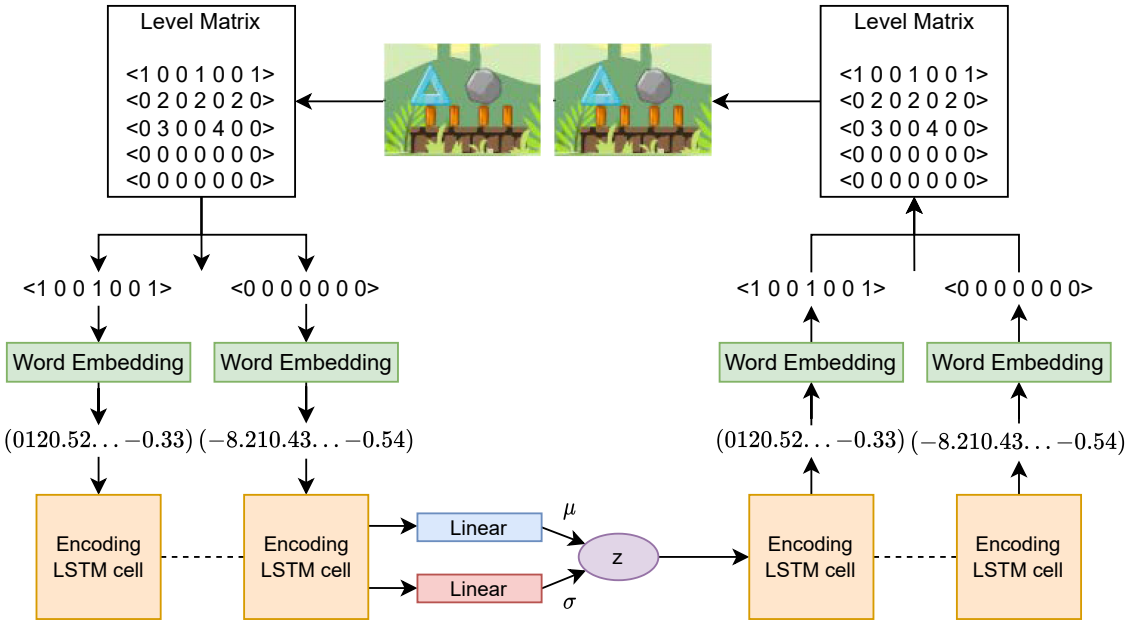

Figure 2.11: Flowchart of the proposed approach by Tanabe et al. 2021. The chart is a recreation based on their provided Flowchart.

Figure 2.11 describes how a level is encoded into a level matrix. The level matrix encodes a level as sequence data capturing the level as if each block is dropped individually from the top on to the next block or platform below. The matrix is then processed as if it were a natural language sentence, with each row representing a word. The word embedding is put into the VAE-LSTM model with the training goal to of recreating the level. Creating new levels is archived through LVE in between the autoencoders encoder and decoder.

3 Concepts

4 Approach

Tr

5 Evaluation

6 Discussion & Conclusion

List of Figures

Bibliography

Amato, Alba. 2017. "Procedural Content Generation in the Game Industry." In *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation,* edited by Oliver Korn and Newton Lee, 15–25. Cham: Springer International Publishing. ISBN: 978-3-319-53088-8. https://doi.org/10.1007/978-3-319-53088-8_2.

Arjovsky, Martin, and Léon Bottou. 2017. "Towards Principled Methods for Training Generative Adversarial Networks." *arXiv.org,* https://doi.org/10.48550/arXiv.1701.04862.

Arjovsky, Martin, et al. 2017. *Wasserstein GAN.* https://doi.org/10.48550/ARXIV.1701.07875.

Ba, Jimmy Lei, et al. 2016. *Layer Normalization.* https://doi.org/10.48550/ARXIV.1607.06450.

Basodi, Sunitha, et al. 2020. "Gradient amplification: An efficient way to train deep neural networks." *Big Data Mining and Analytics* 3 (3): 196–207. https://doi.org/10.26599/BDMA.2020.9020004.

Bontrager, Philip, et al. 2017. *DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution.* https://doi.org/10.48550/ARXIV.1705.07386.

Doull, Andrew. 2015. "Procedural Content Generation Wiki Beneath Apple Manor." Accessed August 9, 2022. http://pcg.wikidot.com/pcg-games:beneath-apple-manor.

———. 2016. "Procedural Content Generation Wiki Rogue." Accessed August 9, 2022. http://pcg.wikidot.com/pcg-games:rogue.

Ferreira, Lucas, and Claudio Toledo. 2014. "A Search-based Approach for Generating Angry Birds Levels." In *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games.* CIG'14. Dortmund, Germany.

Fontaine, Matthew C, et al. 2020. "Illuminating Mario Scenes in the Latent Space of a Generative Adversarial Network." *arXiv.org,* https://doi.org/10.48550/arXiv.2007.05674.

Giacomello, Edoardo, et al. 2018. *DOOM Level Generation using Generative Adversarial Networks.* https://doi.org/10.48550/ARXIV.1804.09154.

Goodfellow, Ian. 2017. *NIPS 2016 Tutorial: Generative Adversarial Networks.* https://doi.org/10.48550/ARXIV.1701.00160.

Goodfellow, Ian, et al. 2014. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems,* edited by Z. Ghahramani et al., vol. 27. Curran Associates, Inc.

Goodfellow, Ian, et al. 2016. *Deep Learning.* Http://www.deeplearningbook.org. MIT Press.

Gui, Jie, et al. 2021. "A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications." *IEEE Transactions on Knowledge and Data Engineering,* 1–1. https://doi.org/10.1109/tkde.2021.3130191.

Gulrajani, Ishaan, et al. 2017. "Improved Training of Wasserstein GANs." *CoRR* abs/1704.00028.

Hinton, Geoffrey, et al. n.d. "Neural Networks for Machine Learning Lecture 6a Overview of mini–batch gradient descent."

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-term Memory." *Neural computation* 9 (December): 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.

Hui, Jonathan. 2018. *GAN — Wasserstein GAN & WGAN-GP - Jonathan Hui - Medium,* June.

Ioffe, Sergey, and Christian Szegedy. 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* https://doi.org/10.48550/ARXIV.1502.03167.

Jabbar, Abdul, et al. 2020. "A Survey on Generative Adversarial Networks: Variants, Applications, and Training." *arXiv.org,* https://doi.org/10.48550/arXiv.2006.05132.

Karras, Tero, et al. 2017. *Progressive Growing of GANs for Improved Quality, Stability, and Variation.* https://doi.org/10.48550/ARXIV.1710.10196.

Karras, Tero, et al. 2018. "A Style-Based Generator Architecture for Generative Adversarial Networks." *CoRR* abs/1812.04948.

Kingma, Diederik P, and Max Welling. 2013. *Auto-Encoding Variational Bayes.* https://doi.org/10.48550/ARXIV.1312.6114.

LeCun, Yann. 2016. "What are some recent and potentially upcoming breakthroughs in deep learning?" Accessed August 13, 2022. https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning.

Liu, Jialin, et al. 2020. "Deep learning for procedural content generation." *Neural Computing and Applications* 33, no. 1 (October): 19–37. https://doi.org/10.1007/s00521-020-05383-8.

Liu, Ming-Yu, and Oncel Tuzel. 2016. *Coupled Generative Adversarial Networks.* https://doi.org/10.48550/ARXIV.1606.07536.

Mirza, Mehdi, and Simon Osindero. 2014. *Conditional Generative Adversarial Nets.* https://doi.org/10.48550/ARXIV.1411.1784.

Moghadam, Monireh Mohebbi, et al. 2021. "Game of GANs: Game-Theoretical Models for Generative Adversarial Networks." *arXiv.org,* https://doi.org/10.48550/arXiv.2106.06976.

Nair, Vinod, and Geoffrey E. Hinton. 2010. "Rectified Linear Units Improve Restricted Boltzmann Machines." In *Proceedings of the 27th International Conference on International Conference on Machine Learning,* 807–814. ICML'10. Haifa, Israel: Omnipress. ISBN: 9781605589077.

Radford, Alec, et al. 2015. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* https://doi.org/10.48550/ARXIV.1511.06434.

Salimans, Tim, et al. 2016. *Improved Techniques for Training GANs.* https://doi.org/10.48550/ARXIV.1606.03498.

Saxena, Divya, and Jiannong Cao. 2020. "Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions." *CoRR* abs/2005.00065.

Schmidhuber, Jürgen. 2015. "Deep learning in neural networks: An overview." *Neural Networks* 61 (January): 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.

Silver, David, et al. 2016. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529, no. 7587 (January): 484–489. https://doi.org/10.1038/nature16961.

Smelik, Ruben M., et al. 2014. "A Survey on Procedural Modelling for Virtual Worlds." *Comput. Graph. Forum* (Chichester, GBR) 33, no. 6 (September): 31–50. ISSN: 0167-7055. https://doi.org/10.1111/cgf.12276.

Stephenson, Matthew, and Jochen Renz. 2016a. "Procedural generation of complex stable structures for angry birds levels." In *2016 IEEE Conference on Computational Intelligence and Games (CIG),* 1–8. https://doi.org/10.1109/CIG.2016.7860410.

———. 2016b. "Procedural Generation of Levels for Angry Birds Style Physics Games." In *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.* AIIDE'16. Burlingame, California, USA: AAAI Press. ISBN: 978-1-57735-772-8.

———. 2017. "Generating varied, stable and solvable levels for angry birds style physics games." In *2017 IEEE Conference on Computational Intelligence and Games (CIG),* 288–295. https://doi.org/10.1109/CIG.2017.8080448.

Stephenson, Matthew, et al. 2019. "The 2017 AIBIRDS Level Generation Competition." *IEEE Transactions on Games* 11 (3): 275–284. https://doi.org/10.1109/TG.2018.2854896.

Summerville, Adam, and Michael Mateas. 2016. *Super Mario as a String: Platformer Level Generation Via LSTMs.* https://doi.org/10.48550/ARXIV.1603.00930.

Summerville, Adam, et al. 2017. "Procedural Content Generation via Machine Learning (PCGML)." *arXiv.org,* https://doi.org/10.48550/arXiv.1702.00539.

Summerville, Adam James, et al. 2016. *The VGLC: The Video Game Level Corpus.* https://doi.org/10.48550/ARXIV.1606.07487.

Tanabe, Takumi, et al. 2021. "Level Generation for Angry Birds with Sequential VAE and Latent Variable Evolution." In *Proceedings of the Genetic and Evolutionary Computation Conference,* 1052–1060. GECCO '21. Lille, France: Association for Computing Machinery. ISBN: 9781450383509. https://doi.org/10.1145/3449639.3459290.

Tao, Chenyang, et al. 2018. “Chi-square Generative Adversarial Network.” In *Proceedings of the 35th International Conference on Machine Learning,* edited by Jennifer Dy and Andreas Krause, 80:4887–4896. Proceedings of Machine Learning Research. PMLR, June.

Togelius, Julian, et al. 2013. “The Mario AI Championship 2009-2012.” *AI Magazine* 34, no. 3 (September): 89–92. https://doi.org/10.1609/aimag.v34i3.2492.

Tran, Ngoc-Trung, et al. 2018. *Dist-GAN: An Improved GAN using Distance Constraints.* https://doi.org/10.48550/ARXIV.1803.08887.

Volz, Vanessa, et al. 2018. “Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018).* Kyoto, Japan: ACM, July. https://doi.org/10.1145/3205455.3205517.

Wang, Yaxing, et al. 2016. “Ensembles of Generative Adversarial Networks.” *CoRR* abs/1612.00991.

Xu, Bing, et al. 2015. *Empirical Evaluation of Rectified Activations in Convolutional Network.* https://doi.org/10.48550/ARXIV.1505.00853.

Yannakakis, G. N., and J. Togelius. 2011. “Experience-Driven Procedural Content Generation.” *IEEE Transactions on Affective Computing* 2, no. 3 (July): 147–161. https://doi.org/10.1109/t-affc.2011.6.

Zeiler, Matthew D, and Rob Fergus. 2013. *Visualizing and Understanding Convolutional Networks.* https://doi.org/10.48550/ARXIV.1311.2901.

A Appendix