

Utilizing Generative Adversarial Networks for Stable Structure Generation in Angry Birds

Frederic Abraham¹, Matthew Stephenson²

¹ Maastricht University, Department of Advanced Computing Sciences, Maastricht, the Netherlands

² Flinders University, College of Science and Engineering, Adelaide, Australia
fm.abraham@alumni.maastrichtuniversity.nl, matthew.stephenson@flinders.edu.au

Abstract

This paper investigates the suitability of using Generative Adversarial Networks (GANs) to generate stable structures for the physics-based puzzle game Angry Birds. While previous applications of GANs for level generation have been mostly limited to tile-based representations, this paper explores their suitability for creating stable structures made from multiple smaller blocks. This includes a detailed encoding/decoding process for converting between Angry Birds level descriptions and a suitable grid-based representation, as well as utilizing state-of-the-art GAN architectures and training methods to produce new structure designs. Our results show that GANs can be successfully applied to generate a varied range of complex and stable Angry Birds structures.

Introduction

Procedural Content Generation (PCG), which describes the creation of content through algorithmic means, has become an increasingly prominent aspect of video game development (Amato 2017). In the same timeframe as conventional PCG techniques were being researched, the number of applications utilizing Machine Learning (ML) approaches has also increased, with techniques such as Neural Networks and Deep Learning receiving a large amount of attention (Goodfellow, Bengio, and Courville 2016). Consequently, the use of ML approaches for content generation, under the abbreviation PCGML, has become an area of significant research interest (Summerville et al. 2017).

While many different types of ML algorithms have been used to generate content, one of the most promising approaches in recent years has been the use of Generative Adversarial Networks (GANs) (Gui et al. 2021; Jabbar, Li, and Omar 2021; Saxena and Cao 2020). A GAN is the construct of two adversarial networks, in which one network generates new content and the second network discriminates the generated content to differentiate between real and generated (Goodfellow et al. 2014). While the primary application of GANs has largely been for image and video synthesis (Huang, Yu, and Wang 2018; Liu et al. 2021), they have also been applied to several other domains including video game content generation (Giacomello, Lanzi, and Loiacono 2018; Volz et al. 2018). However, to the best of our knowledge,

when it comes to generating level-based content for video games, GANs have only been applied to discrete domains with no physical constraints.

In this paper, we present an approach for generating stable structures made of several rectangular blocks in a continuous physical environment. More specifically, we generate structures for use in the 2D physics-based puzzle game Angry Birds (Rovio 2023). This domain provides a unique and novel challenge for applying GAN-based approaches, where considerations such as the game’s continuous environment space and physical constraints must be taken into account. Our results demonstrate that such an approach is viable for generating complete Angry Birds structures.

The remainder of this paper is organized as follows. We first describe background details and prior work related to GANs and their use in level generation, as well as previous approaches to Angry Birds level generation. Next, we describe the methodology we employed with regards to level encoding and decoding, as well as our GAN model training process. We then describe our experiments, providing details on how our specific GAN model was trained and analyze the generated levels it was able to produce. We finish with a summary conclusion of our approach and output, along with suggestions for future work.

Background and Related Work

Generative Adversarial Networks

A GAN is a framework for producing generative models via a process in which two networks, the generator and the discriminator, are trained simultaneously (Goodfellow et al. 2014). The generator aims to capture the data distribution of the training data, while the discriminator, also called the critic, 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 central concept behind applying GANs is to define the given task as a game between two opposing systems, which are then trained in an adversarial manner to reach a zero-sum Nash equilibrium (Moghadam et al. 2021). While GANs have been primarily used for image synthesis (Goodfellow et al. 2014; Radford, Metz, and Chintala 2016; Liu and Tuzel 2016; Karras et al.

2017; Huang, Yu, and Wang 2018), they have also been successfully applied to many other applications including object detection (Prakash and Karam 2021; Posilović et al. 2021), natural language processing (Subramanian et al. 2017), audio enhancement (Torres-Reyes and Latifi 2019; Biswas and Jia 2020), anomaly detection (Schlegl et al. 2017; Xia et al. 2020), and, most relevant for this paper, video game level generation.

In recent years, several variations and improvements to GANs have been proposed for different aspects of the underlying framework. The architecture of each network, the controllability, the training process, scaling, adaptation and application in different domains are a few research directions that have been investigated. Some notable advancements include the development of Deep Convolutional GANs (DCGANs) (Radford, Metz, and Chintala 2016), which introduced deeper architectures and improved training stability for image synthesis tasks. Another significant improvement came with the introduction of Wasserstein GANs (WGANs) (Arjovsky, Chintala, and Bottou 2017), which proposed a new objective function to address training difficulties and mode collapse issues (Salimans et al. 2016). The term mode collapse refers to the situation where a GAN model repeatedly generates highly similar outputs that don't represent the variety of content present in the original training set.

Level Generation with GANs

While GANs have been used to generate a variety of different types of game content, such as the generation of NPC character sprites (Kim et al. 2023; Coutinho and Chaimowicz 2022), we will focus primarily on the use of GANs to generate game levels. One of the first to apply GANs in the context of level generation was Giacomello et al., in their work to generate DOOM level images based on human-designed examples (Giacomello, Lanzi, and Loiacono 2018). They describe their preliminary results as a good starting point for researching the viability of GANs compared to classical PCG. The generated level images contained DOOM typical features and are reportedly interesting to explore, although the generated data could not be decoded into playable levels.

Volz et al. also utilized GANs to generate complete Mario levels (Volz et al. 2018). They use a 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. Their 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 first introduced by Bontrager et al., in their works to match generated fingerprints to as many real fingerprints as possible (Bontrager, Togelius, and Memon 2017). 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 com-

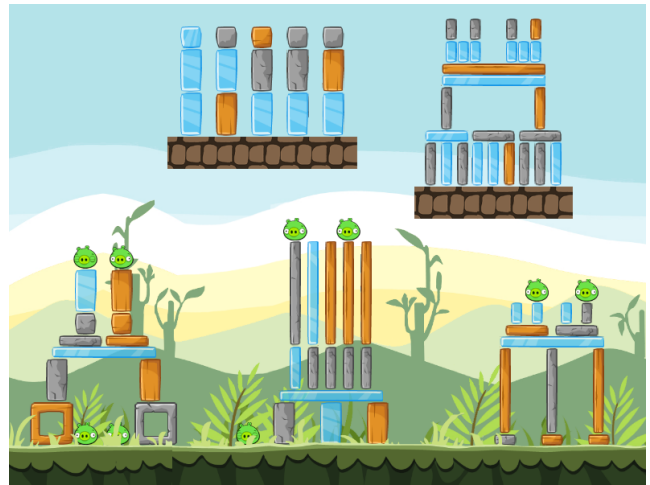


Figure 1: An example Angry Birds level containing five distinct structures.

Id	Shape	Name	Dimensions
1		SquareHole	(0.85, 0.85)
2		RectBig	(2.06, 0.22)
3		RectMedium	(1.68, 0.22)
4		RectSmall	(0.85, 0.2)
5		RectFat	(0.85, 0.43)
6		RectTiny	(0.42, 0.22)
7		SquareTiny	(0.22, 0.22)
8		SquareSmall	(0.43, 0.43)

Table 1: Block types that are available in Science Birds.

binated with the noise vector as input to the generator to produce a controllable output. They conclude that GANs can capture high-level structures of the training level, although they may sometimes produce broken elements such as incomplete pipes and structures.

Angry Birds Level Generation

Angry Birds is a 2D physics-based simulation game, where players are tasked with shooting birds at structures made of multiple smaller blocks. These blocks are each made of a specific material (wood, ice or stone) which affects their resistance to certain bird types. The players objective in each level is to kill all of the pigs, which are often located within or on top of structures, using a limited number of birds. An example Angry Birds level, containing five structures and nine pigs, is shown in Figure 1. Due to the fact that the original Angry Birds game is not open source, most research on Angry Birds instead uses a Unity-based clone of the game called Science Birds, created by Lucas Ferreira (Ferreira and Toledo 2014). This clone contains all the same elements as the original Angry Birds game, and each level is described in a single xml file containing the position of each object.

Table 1 shows the available block types that structures can be made from, along with their names and dimensions (width, height). While there are several additional “irregular” block shapes that are also available in Angry Birds, specifically two triangular and two circular block types, the vast majority of Angry Birds structures do not include these. Many prior structure generators for Angry Birds either excluded irregular blocks entirely or used them in a purely decorative manner, as the stability of a structure is much harder to verify if irregular blocks are present. In addition to this, blocks are almost always placed at 90-degree angles, once again to reduce the risk of structural instability. As a result of this, our proposed GAN-based generator will be solely trained on, and will therefore only output, structures made of the block types specified in Table 1, placed in either a horizontal (angle = 0) or vertical (angle = 90) orientation.

Over the past decade a large number of different level generators for Angry Birds have been proposed, with many of them being entered into the AIBirds Level generation competition (Stephenson et al. 2019). These generators have experimented with a variety of different approaches, including Genetic Algorithms (Ferreira and Toledo 2014), search-based techniques (Stephenson and Renz 2016a,b, 2017), Monte Carlo Tree Search (Graves, Caramanis, and Nagarajan 2016), and latent variable evolution in a variational auto encoder (Tanabe et al. 2021), as well as focusing on a variety of different desirable aspects, such as levels that contain structures which resemble quotes or formulas (Jiang, Harada, and Thawonmas 2017), deceptive elements (Gamage et al. 2021), or Rube Goldberg Machine mechanisms (Abdullah et al. 2019).

Methodology

This section describes the different components that make up our proposed GAN framework for Angry Birds levels generation. This includes level encoding, level decoding and GAN Model Training.

Level Encoding

This section describes the encoding process that was used to convert an XML level description of a structure into a suitable grid-based data representation that can be used to train our proposed GAN model.

Grid-Based Representation One of the major challenges in this domain is the real-valued positioning and dimensions of in-game objects. Continuous values such as these are impractical for GAN models, which typically require the input to be in a discrete, grid-based representation to function effectively (e.g., the pixels of an image, or the individual tiles in a Mario level). As a result, our XML structure representation first needs to be converted or encoded to a grid-based representation with a specified degree of precision (i.e., a raster size). Figure 2 shows an example of a simple Angry Birds structure with continuous block positions and dimensions, represented visually as a wireframe diagram with different colors for each block material and pigs, that will be used as a reference throughout this section.

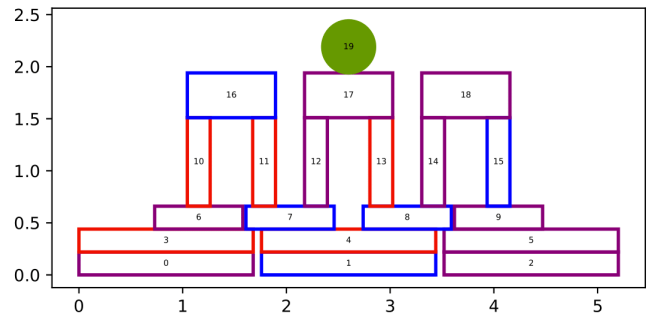


Figure 2: An example Angry Birds structure represented as a colored wireframe diagram.

In order to discretize a structure to a specified level of precision, a suitable raster size has to be chosen. A smaller raster size better represents the blocks true dimensions, and is able to capture small gaps and imperfect positionings that a larger raster size may miss. The downside of a smaller raster size is that it can dramatically increase the dimensionality of the encoded structure representation, requiring a larger GAN model that results in significantly longer training and generation times. Conversely, a bigger raster size results in a smaller encoded output representation where only a few grid cells represent any given block. This comes with a loss of finer details, that may potentially result in incorrectly represented block sizes.

As a suitable compromise between these two considerations, we selected the largest raster size that resulted in a near-integer value when dividing by any block’s true dimension. This was done to reduce the risk of the same block type being encoded at different sizes, depending on its relative position to the grid lines. For the block sizes present in ScienceBirds a raster size 0.07 resulted in quotients with only a small remainder, and was subsequently chosen as our raster size. This raster size of 0.07 in-game distance units, can be equated to 1 unit of our proposed grid dimension encoding.

Using this chosen raster size, we can now convert any given XML level description into a matching grid-based encoding. For each block, the horizontal and vertical start and end positions are calculated by taking the given center position of the block and adding/subtracting half of the width/height respectively. To transform these positions into the grid indices, each value is divided by the raster size (0.07) and rounded to the nearest integer. Doing this for each object has the effect of converting its position and size to exactly fit our defined grid dimensions. Figure 3 shows how the example structure shown in Figure 2 can be converted using this proposed encoding approach into a rasterized image.

One limitation with this structure encoding approach is that number and size of the grid cells is fixed, which places a bound on the maximum structure size that could be encoded. For our GAN model, we fixed our grid dimension at 128x128 cells or approximately 9x9 in-game distance units. This can be increased to support larger structures if desired, although this will naturally result in longer training and generation times.

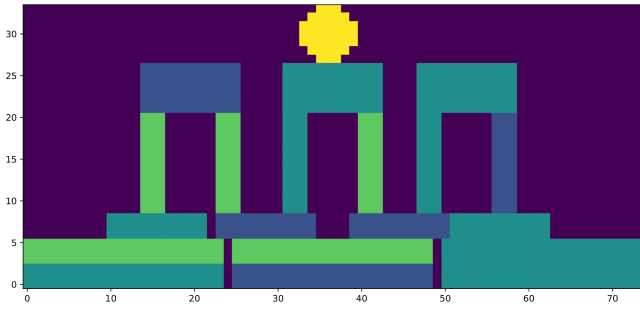


Figure 3: An encoded (rasterized) 2D image of an Angry Birds structure to a precision (resolution) of 0.07.

Multilayer Representation While it is possible to represent all four object types (wood/ice/stone blocks and pigs) as different values (wood = 1, ice = 2, etc.) within a single 2D grid representation, this makes training our GAN model more difficult. With this representation approach, our GAN model must simultaneously decide about the positioning of the elements and their material within a single layer. Looking at prior work, GAN data representations are usually not only two-dimensional. For example, face synthesis requires three channels for the RGB-Color space, and Volz et al. (Volz et al. 2018) used ten channels in a one-hot encoding, one for each Mario block type. By moving each object type to separate layers, the decision for each layer is simplified to only predicting if an element of the associated type is present. Figure 4 shows a visual representation of this multilayer encoding approach for the same 2D image shown in Figure 3. Using this multilayer representation splits the blocks of each material type into separate layers, with pigs and air (i.e., empty space) each getting their own layer as well, making for a total of five layers.

Level Decoding

This section describes the inverse of the previous process, that of taking a structure that is output by our trained GAN model using our specified grid-based representation and decoding it into a valid XML level description for Science Birds.

Confidence Decoding Most GAN-based approaches for procedural content generation use an encoding method that allows for a one-to-one decoding, meaning that no extra steps are required to create the level. This can potentially lead to broken elements, such as a pipe that may be missing several tiles (Volz et al. 2018) and can be solved by encoding the whole group of tiles together. The decoding process can also include a healing section, where a post-processing algorithm fixes errors, tests for playability and repairs the level accordingly.

When using the multilayered encoding defined in the previous section our GAN outputs a real-valued matrix for each object type, with each matrix having the same dimensions as our defined grid representation. A value closer to one for a given matrix entry represents a higher confidence that an object of that type should be present in the corresponding grid

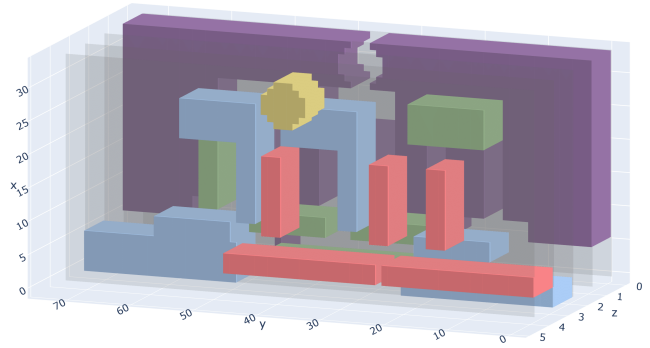


Figure 4: A multilayer representation of an Angry Birds structure, with separate layers for each object type visualized along the z-axis

cell, with a value close to one for the air layer representing that there should be no object present. In other words, our output matrices represent the predicted confidence that each GAN layer has for an object of its type being present at any given grid cell. This multilayer representation can be subsequently converted back into a flat 2D image using the argmax operator, where the layer with the highest confidence value output is used. However, simply determining the object type for each grid cell will not be sufficient, as our output XML representation can only include blocks of predefined sizes. In other words, each block is encoded into, and must also therefore be decoded from, multiple grid cells.

Matrix Creation To decode a generated structure image back into a valid xml level description, we need to identify suitable block positions for each material that maintain the structure’s overall design. The first step of our proposed process involves the creation of a 3-dimensional Selection-Ranking matrix, with dimensions $128 \times 128 \times 13$. Each layer of this matrix represents a specific block type, including separate layers for horizontal and vertical orientations of rectangular block shapes, giving a total of 13 layers. Each of these layers contains 128×128 values representing each position of our encoded grid space, with the value at each position indicating its suitability for placing a block of the associated type. This Selection-Ranking matrix is a combination of two sub-matrices, called the Hit-Probabilities matrix and the Size-Ranking matrix.

The Hit-Probabilities matrix for each layer is created by applying a Gaussian kernel in the shape of the associated block type across the encoded structure representation, see Table 1. Each layer of this matrix represents how well the associated block type would match the encoded structure representation if placed at each possible position (i.e., what is the percentage of overlap between the block and the structure). However, using this matrix alone leads to an issue, namely that smaller blocks (shown in the lower rows) will have a much higher hit probability than larger blocks, simply due to their smaller size. If only the highest hit probabilities were used in the selection process, any small imperfections in the representation would dismiss any larger block types that may have been intended to be there, and a group

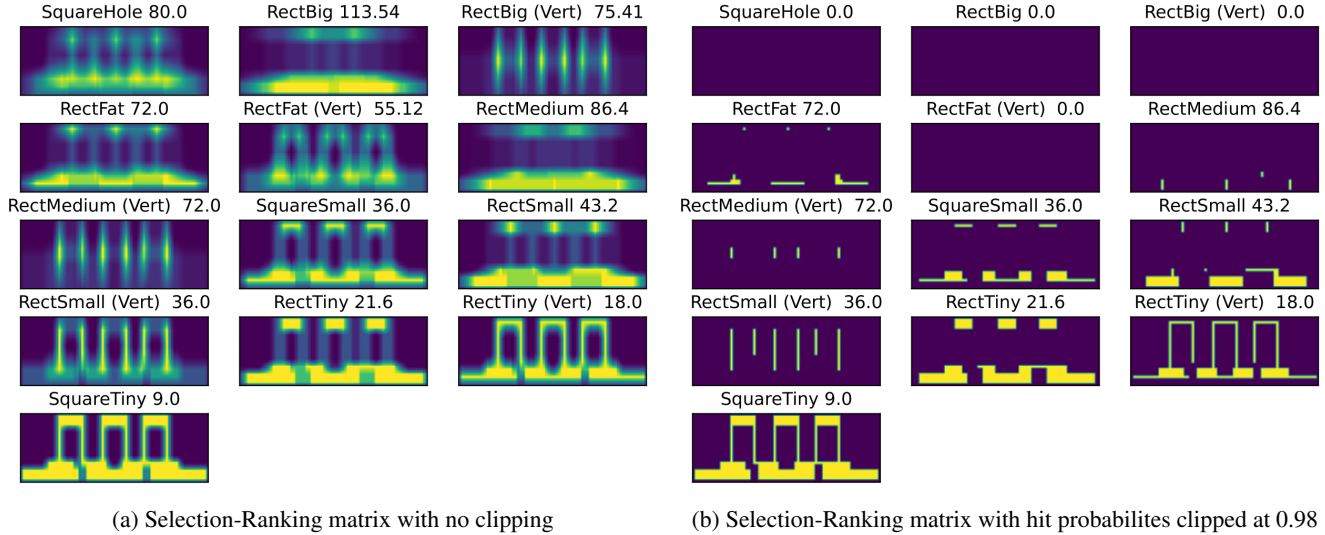


Figure 5: Heatmap representations of the Selection-Ranking matrix for each block type, calculated based on the layer-wise multiplication of the Hit-Probability matrix and Size-Ranking matrix. Subfigure (a) represents the original Selection-Ranking matrix; while subfigure (b) represents the same Selection-Ranking matrix but with the Hit-Probabilities matrix clipped at 0.98. Each image represents a specific block type and corresponds to an individual layer of the Selection-Ranking matrix. The value above each image represents the maximum value present in this layer.

of smaller blocks would likely be used instead. To address this, we introduced an additional Size-Ranking matrix that applies a sum kernel to add up the values of all covered pixels for each block type at each location.

Using the encoded Angry Birds structure representation from Figure 3 as our input, Figure 5a shows the layer-wise multiplication of these two sub-matrices into the combined Selection-Ranking matrix, where yellow areas represent locations where placing the associated block type would most match the encoded structure representation. It can be seen in Figure 5a that the horizontal RectBig layer has the highest selection value (113.54), even though it crosses gaps in the encoded structure image. By clipping our Hit-Probabilities matrix at a high value, in this case 0.98, any block types that cross gaps such as these are removed. In practice, this has the effect of making our Selection-Ranking matrix more distinct with harder edges, see Figure 5b, and ensures that any small gaps in the original structure encoding are more likely to be maintained in the decoded output. This final clipped version of the Selection-Ranking matrix is then used as input for the subsequent block selection algorithm.

Block Selection Our block selection algorithm starts by identifying the highest value in our clipped Selection-Ranking matrix, which corresponds to the “best fit” location for a specific block type, see blue block outline for the horizontal RectMedium layer in Figure 6a. We then place a block of this type at this location and set any values in our matrix that would overlap this block to zero, see red outlines for the other block types/layers in Figure 6a. This process repeats until all values in our matrix are set to zero. The clipped Selection-Ranking matrix shown in Figure 5b

takes a total of 18 iterations for this process to finish, where in each iteration a new block is selected and added to the final structure output. Figure 6 shows several iterations of the Selection-Ranking matrix throughout this block selection process. We then use a circular kernel to identify possible pig positions, followed by the same location selection process.

The last step of the decoding process is to address the issue of blocks slightly overlapping each other, which can occur when a block has an uneven width/height. We therefore apply a small structure adjustment that moves each block up and to the right, until it no longer overlaps any blocks that are below it or on its left side.

The finished result of this decoding process can be seen in Figure 7. By comparing this to the original structure representation in Figure 2 we can see that there are some minor differences between them, but the overall shape and design of the structure is sufficiently similar. This example demonstrates that our proposed encoding and decoding process is able to successfully convert between playable XML level descriptions, and grid-based structure representations that are appropriate for GAN training.

GAN Training

With the encoding and decoding process now defined, this section describes how our encoded structure representation is processed and used for training our GAN model. In order to train our GAN model, two aspects need to be selected. Firstly, the specific architecture that will be used for the generator and discriminator networks. Secondly, the objective function that will be used to update the weights in these networks towards a desirable output.

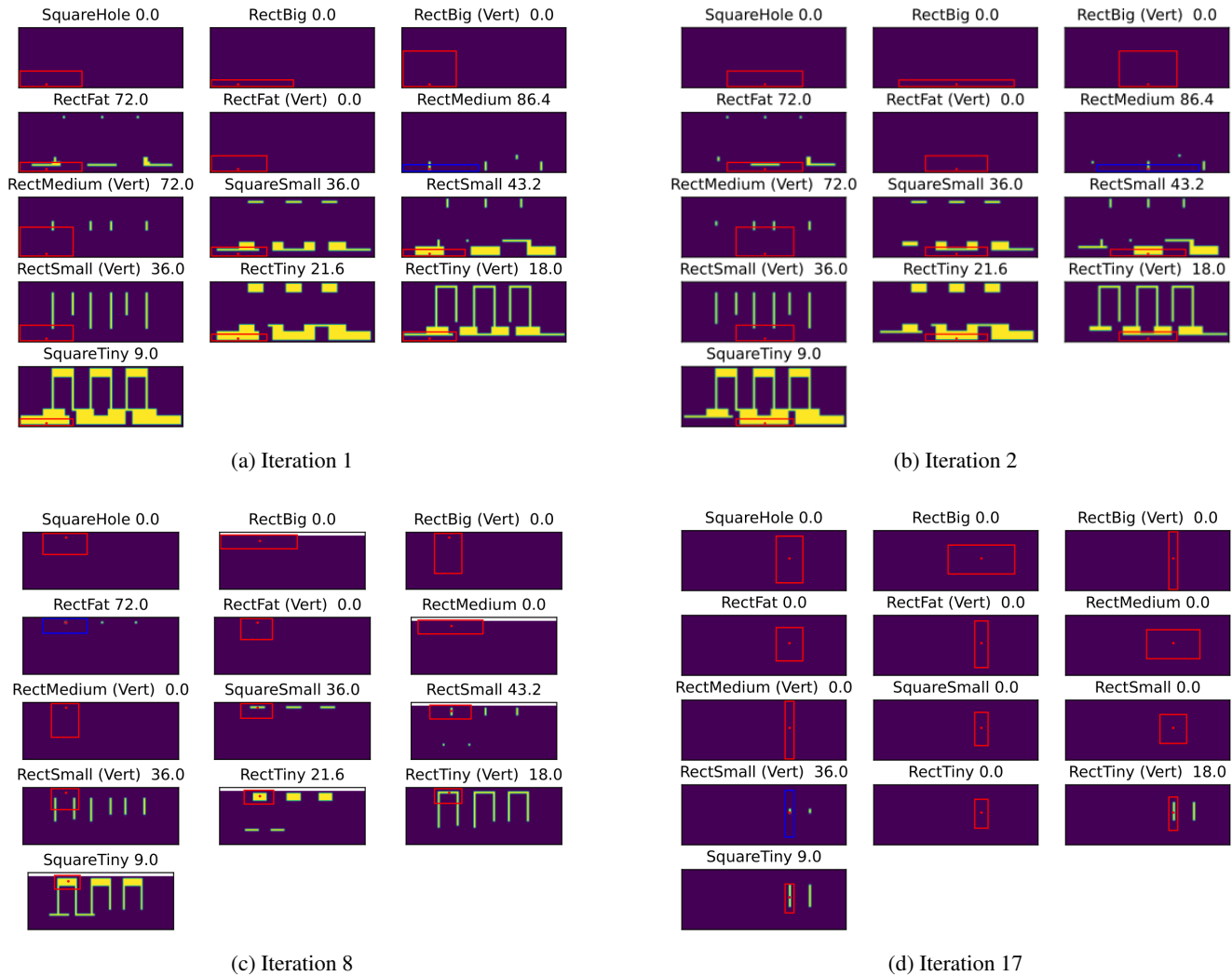


Figure 6: Heatmap representations of the clipped Selection-Ranking matrix for iterations 1, 2, 8 and 17 of the block selection process. The block type (matrix layer) and location with the highest Selection-Ranking matrix value is chosen at the start of each iteration (represented by the blue block outlines). Any values/positions in the Selection-Ranking matrix that are no longer valid, due to overlapping this selected block, are set to zero for future iterations (represented by the red outlines). This process repeats until all values in the Selection-Ranking matrix are equal to zero.

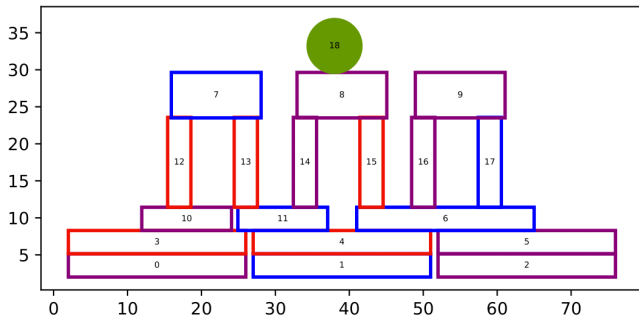


Figure 7: The result of the decoding process.

GAN Architecture Our proposed GAN model is based on the Deep Convolutional Generative Adversarial Network (DCGAN) architecture (Radford, Metz, and Chintala 2016). This is the first Convolutional Neural Network based GAN architecture and has been shown to perform well in image generation tasks (Jabbar, Li, and Omar 2021). Compared to a more standard GAN architecture, which only used fully connected layers and pooling layers, the proposed DCGAN architecture instead uses transposed convolutions to create the required image size.

The generator network portion of our proposed GAN is shown in Figure 8. This network exclusively uses transposed convolutional layers to enlarge the image, with a stride value of two effectively doubling the resolution size with each

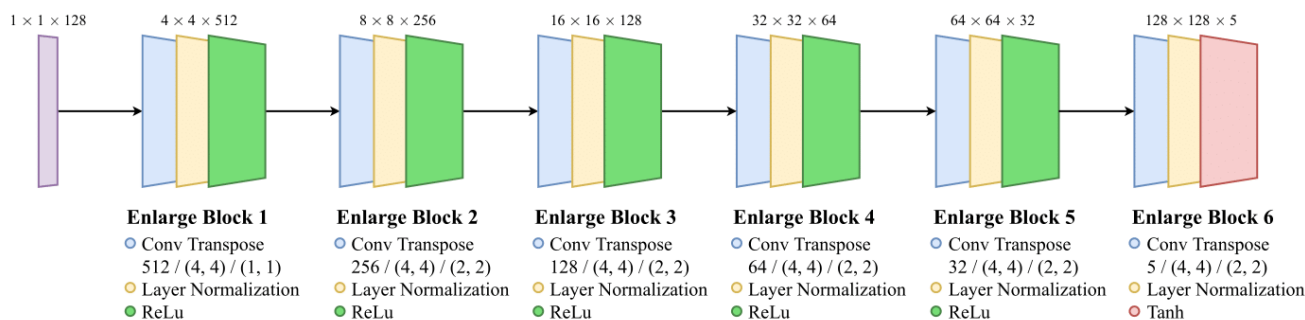


Figure 8: Generator network architecture.

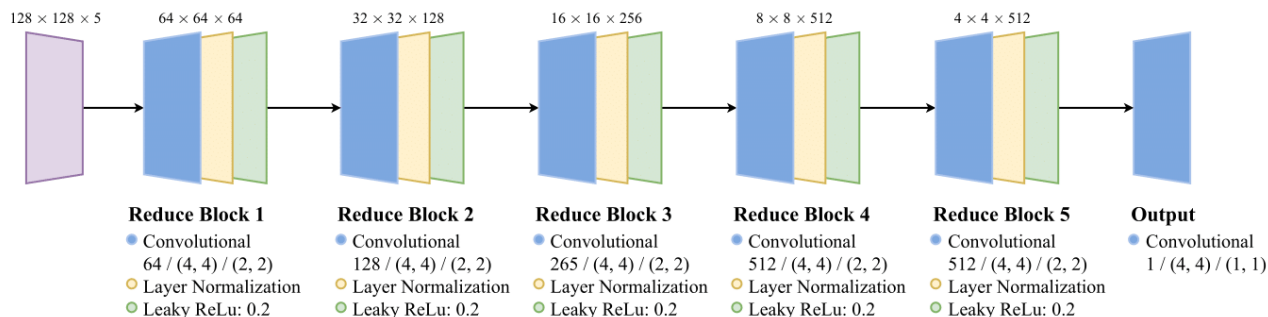


Figure 9: Discriminator network architecture.

layer block, as this has been shown to significantly improve the stability of GAN training (Jabbar, Li, and Omar 2021). The standard batch normalization layers were instead replaced by layer normalization (Ba, Kiros, and Hinton 2016), to reduce the risk of mode collapse during training. The generator uses the ReLU activation function (Nair and Hinton 2010) between the layers and a Tanh function at the output layer, similar to the original implementation by Goodfellow et al. (Goodfellow et al. 2014).

The discriminator network is visualized in Figure 9. While the generator uses transposed convolutional layers, the discriminator uses convolutional layers to reduce the image size and arrive at a decision. The discriminator uses a Leaky ReLU activation function (Xu et al. 2015), based on the recommendation by Radford et al. in their architecture guidelines for stable Deep Convolutional GANs (Radford, Metz, and Chintala 2016).

Objective Function For our objective function, we decided to use the Wasserstein objective function that is based on the Wasserstein or Earth Mover’s distance metric (Arjovsky, Chintala, and Bottou 2017). This approach provides several benefits, including increased training stability and reduced risk of mode collapse. In addition, we use an updated version of this objective function that applies a gradient penalty to the discriminator network (Gulrajani et al. 2017), rather than the regular weight clipping approach. This change has been shown to further increase GAN training stability and convergence likelihood.

Experiments

Our experiments consisted of training the proposed GAN model on an example set of encoded levels descriptions, as well as evaluating the quality and variety of output structures that it was able to produce. The complete set of all generated levels, as well as our fully trained GAN model, is open access and can be downloaded from a public GitHub repository¹.

Model Training

Due to the limited number of human-made levels that are currently available for Science Birds, we decided to use the open-source level generator Iratus Aves (Stephenson and Renz 2017) to provide structures for our training dataset. Using this generator, we created an initial training dataset of 5000 XML level descriptions that each define a single structure (including pigs).

One important consideration when training GANs is to ensure that the training dataset is sufficiently balanced in terms of its content diversity, as this can reduce the risk of mode collapse occurring. To achieve this, we applied a filter to remove any overly similar structures from our training dataset, which in turn would increase the average structure diversity. We first applied a metadata filter to remove any structures with the same number of blocks for each material, as well as the same width and height with a 0.1 unit margin. We next applied a shape filter to remove any struc-

¹<https://github.com/Blaxtzer/Utilizing-Generative-Adversarial-Networks-for-Stable-Structure-Generation-in-Angry-Birds>

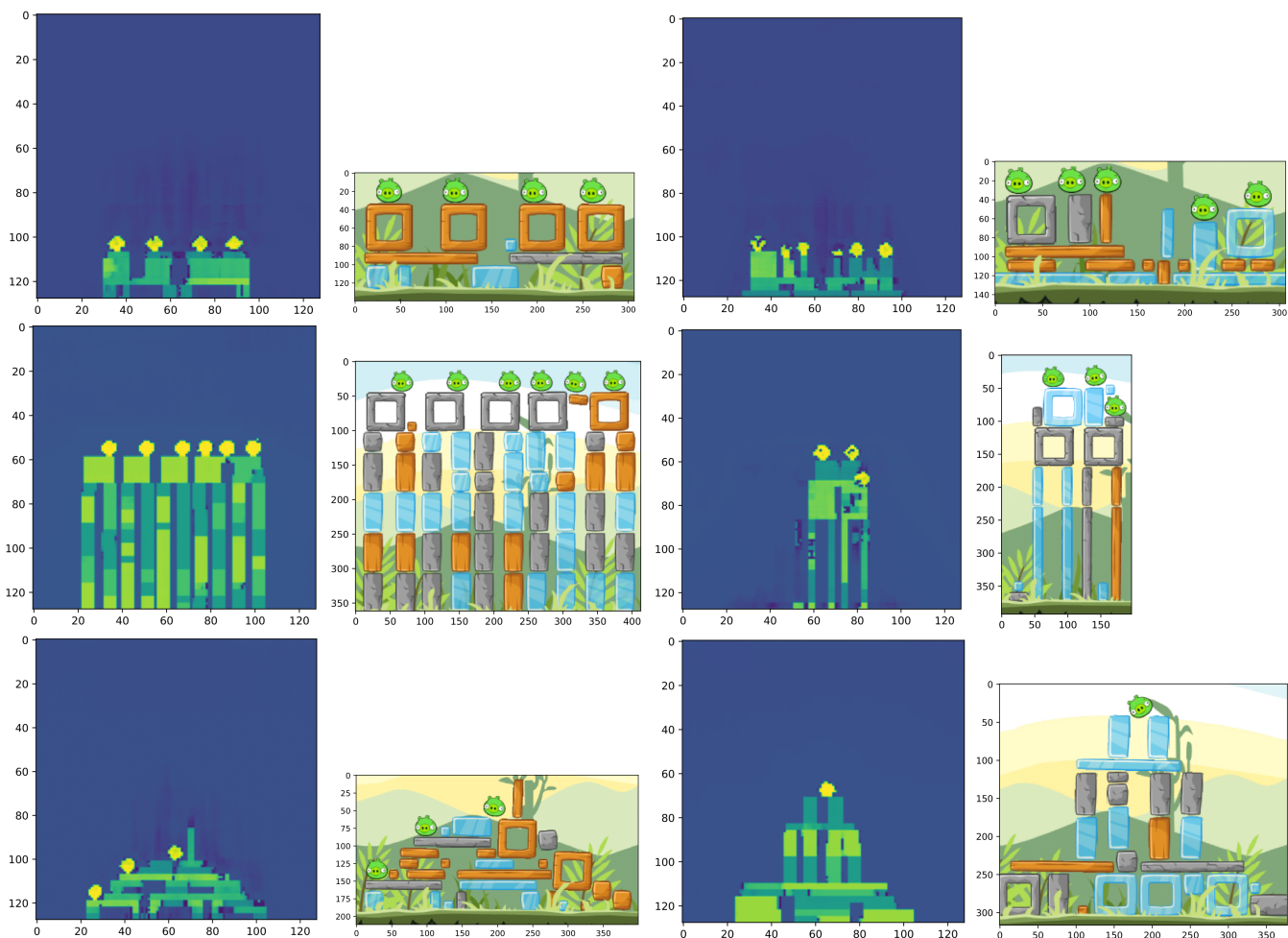


Figure 10: Example GAN outputs and corresponding decoded Angry Birds structures.

tures with the same encoding outline, regardless of block shape or material type. After applying both these filters, our training dataset size was reduced from 5000 to 3566.

Our GAN model was trained for 15000 epochs on our filtered training dataset of 3566 generated structures. This was performed using the RWTH High-Performance Computing cluster (Aachen 2023) and took approximately 48 hours to complete.

Results

Our trained GAN model was used to generate 8000 different structure representations, which were then decoded into playable XML level descriptions. This process took around 15 minutes on a desktop computer with an Intel i7-5820k CPU, with over 95% of that time being used for structure decoding. Figure 10 displays several of these GAN generated structure representation outputs, along with their decoded XML level descriptions loaded into Science Birds. Based on these examples, we can see that many of the GAN outputs contained small distortions or noise that makes the type or shape of certain blocks ambiguous. Despite this, our proposed decoding approach can typically handle these mi-

nor imperfections and is able to produce reasonably accurate and complete structures for Angry Birds.

Structure Stability The first test we conducted was to investigate how many of our 8000 generated structures were stable when loaded into the Science Birds game engine. The stability of any given structure can be determined using one of two approaches. The “Block Velocity” measure determines that a structure is stable if all blocks are stationary when the level is loaded. The “Block Destruction” measure determines that a structure is stable if no blocks are destroyed after the level is loaded. Blocks in Angry Birds will typically be destroyed if they fall from a sufficient height or collide with other blocks, meaning that this measure of stability is a good test for if a structure has collapsed.

Using the Block Velocity measure, 945 structures would be classified as stable and 7055 as unstable. Using the Block Destruction measure, 3487 structures would be classified as stable and 4533 as unstable. Please note, that the Block Destruction measure is a strictly weaker version of the Block Velocity measure (i.e., any structure which is classified as stable by the Block Velocity measure is also always classi-

Id	Name	Frequency
1	SquareHole	8.75% ($\pm 5.89\%$)
2h	RectBig	5.89% ($\pm 3.20\%$)
2v	RectBig (Vert)	13.46% ($\pm 10.21\%$)
3h	RectMedium	4.14% ($\pm 2.13\%$)
3v	RectMedium (Vert)	5.91% ($\pm 4.26\%$)
4h	RectSmall	4.18% ($\pm 2.05\%$)
4v	RectSmall (Vert)	7.33% ($\pm 5.24\%$)
5h	RectFat	5.11% ($\pm 2.81\%$)
5v	RectFat (Vert)	15.81% ($\pm 16.84\%$)
6h	RectTiny	6.84% ($\pm 4.12\%$)
6v	RectTiny (Vert)	6.99% ($\pm 4.90\%$)
7	SquareTiny	9.04% ($\pm 5.95\%$)
8	SquareSmall	6.55% ($\pm 4.56\%$)

Table 2: Average frequency of each block type across all 8000 generated structures (\pm SD).

fied as stable by the Block Destruction measure). This large disparity would indicate that, while our generated structures often contain blocks that move slightly after loading, they are much less likely to collapse completely.

Structure Diversity With regards to structure diversity, our generated structures varied significantly in terms of their width, height, density, shape and block frequency. Across all 8000 generated structures, the average width was 4.87 (± 0.87) and the average height was 3.67 (± 1.36). The average density of each structure, calculated as the percentage of the available level space that is occupied by an object, was 37.05% ($\pm 10.56\%$). The average number of blocks was 24.01 (± 10.18), with an exact break down by block type and orientation provided in Table 2. For reference, please refer to Table 1 for the dimensions of each block type. The average number of pigs in each structure was 2.76 (± 1.76), with 9.1% of generated structures containing zero pigs. While this may initially seem like a serious problem, given that a level with no pigs is already solved, it is important to remember that our proposed approach is intended to create single structures rather than complete levels. Structures without pigs can therefore still be included within a level, as long as they are placed alongside one or more structures that do contain pigs. There was also very little difference between any of these values when comparing stable and unstable structures separately.

In terms of creating new and novel structure designs, while our generated structures appear to have several similar design elements to those present in the original training dataset, which is to be expected of a GAN based approach, they also have many differences. For example, the structures produced by the Iratus Aves generator are created using rows of blocks with the same height, leading to highly symmetrical designs (Stephenson and Renz 2017). However, this was not the case for many of our GAN generated structures, such as those shown in the bottom left and top right examples for Figure 10.

Conclusion

In this paper we have presented, implemented, trained, and evaluated a framework for using GANs to generate new structures for Angry Birds. One of the main contributions of this paper is the proposed encoding and decoding process, that can accurately convert between a playable XML level description and a grid-based structure representation more suited for GANs. Using this in conjunction with state-of-the-art GAN architectures, we were able to successfully train a GAN model to produce complete and highly varied structure designs. While many of these generated structures were initially stable, some of them unfortunately collapsed when loaded into our simulation engine. However, the number of unstable structures was not overwhelmingly large, and such structures could easily be discarded after generation via a simple generate-and-test approach. As such, we believe that the use of GAN models to generate Angry Birds structures may be used to provide an abundance of both new training content for AI agents and gameplay experiences for human players.

We would also like to highlight that this paper is a condensed version of an original Masters thesis (Abraham 2022). This paper describes the most successful approach from this thesis, but several alternative encoding/decoding approaches were also investigated. We would encourage interested readers to take a closer look at this thesis for more details.

In terms of future work, one of the first extensions we might make is to train our proposed model on a larger range of content. Our experimental training set contained 3566 structures from a single generator, but there are over a dozen different level generators for Angry Birds that have been proposed during the past decade. Utilizing several of these generators could provide a much more diverse set of training levels, and potentially a more varied range of output structures. Beyond this, we could also experiment with different GAN architectures, stable diffusion models, data representations, or improvements to our encoding/decoding processes. Lastly, this approach could be applied to other physics-based domains beyond Angry Birds.

References

- Aachen, R. 2023. RWTH High Performance Computing. <https://help.itc.rwth-aachen.de/en/service/t4ibytwt1n3p/>. Accessed: 2023-08-16.
- Abdullah, F.; Paliyawan, P.; Thawonmas, R.; Harada, T.; and Bachtiar, F. A. 2019. An Angry Birds Level Generator with Rube Goldberg Machine Mechanisms. In *2019 IEEE Conference on Games (CoG)*, 1–8.
- Abraham, F. M. 2022. *Utilizing generative adversarial networks for stable structure generation in a physics-based simulation*. Master’s thesis, Maastricht University.
- Amato, A. 2017. *Procedural Content Generation in the Game Industry*, 15–25. Cham: Springer International Publishing. ISBN 978-3-319-53088-8.
- 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.
- Ba, L. J.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *CoRR*, abs/1607.06450.
- Biswas, A.; and Jia, D. 2020. Audio Codec Enhancement with Generative Adversarial Networks. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 356–360.
- Bontrager, P.; Togelius, J.; and Memon, N. D. 2017. DeepMasterPrint: Generating Fingerprints for Presentation Attacks. *CoRR*, abs/1705.07386.
- Coutinho, F.; and Chaimowicz, L. 2022. On the Challenges of Generating Pixel Art Character Sprites Using GANs. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 18(1): 87–94.
- Ferreira, L.; and Toledo, C. 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*.
- Gamage, C.; Pinto, V.; Renz, J.; and Stephenson, M. 2021. Deceptive Level Generation for Angry Birds. In *2021 IEEE Conference on Games (CoG)*, 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.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27.
- Graves, M.; Caramanis, C.; and Nagarajan, S. 2016. *Procedural content generation of Angry Birds levels using Monte Carlo Tree Search*. Master's thesis, The University of Texas at Austin.
- Gui, J.; Sun, Z.; Wen, Y.; Tao, D.; and Ye, J. 2021. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *IEEE Transactions on Knowledge and Data Engineering*.
- Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. C. 2017. Improved Training of Wasserstein GANs. *CoRR*, abs/1704.00028.
- Huang, H.; Yu, P. S.; and Wang, C. 2018. An Introduction to Image Synthesis with Generative Adversarial Nets. *CoRR*, abs/1803.04469.
- Jabbar, A.; Li, X.; and Omar, B. 2021. A Survey on Generative Adversarial Networks: Variants, Applications, and Training. *ACM Comput. Surv.*, 54(8).
- Jiang, Y.; Harada, T.; and Thawonmas, R. 2017. Procedural generation of angry birds fun levels using pattern-struct and preset-model. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 154–161.
- Karras, T.; Aila, T.; Laine, S.; and Lehtinen, J. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *CoRR*, abs/1710.10196.
- Kim, J.; Jin, K.; Jang, S.; Kang, S.; and Kim, Y. 2023. Game effect sprite generation with minimal data via conditional GAN. *Expert Systems with Applications*, 211: 118491.
- Liu, M.-Y.; Huang, X.; Yu, J.; Wang, T.-C.; and Mallya, A. 2021. Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications. *Proceedings of the IEEE*, PP: 1–24.
- Liu, M.-Y.; and Tuzel, O. 2016. Coupled Generative Adversarial Networks. In *Advances in Neural Information Processing Systems*, volume 29.
- Mirza, M.; and Osindero, S. 2014. Conditional Generative Adversarial Nets. *CoRR*, abs/1411.1784.
- Moghadam, M. M.; Boroumand, B.; Jalali, M.; Zareian, A.; Javad, A. D.; and Manshaei, M. H. 2021. Game of GANs: Game Theoretical Models for Generative Adversarial Networks. *CoRR*, abs/2106.06976.
- Nair, V.; and Hinton, G. E. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10*, 807–814. Madison, WI, USA: Omnipress. ISBN 9781605589077.
- Posilović, L.; Medak, D.; Subašić, M.; Budimir, M.; and Lončarić, S. 2021. Generative adversarial network with object detector discriminator for enhanced defect detection on ultrasonic B-scans. *Neurocomputing*, 459: 361–369.
- Prakash, C.; and Karam, L. 2021. It GAN DO Better: GAN-based Detection of Objects on Images with Varying Quality. *IEEE Transactions on Image Processing*.
- Radford, A.; Metz, L.; and Chintala, S. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR*.
- Rovio. 2023. Angry Birds. <https://www.angrybirds.com/games/rovio-classics-angry-birds/>. Accessed: 2023-08-16.
- Salimans, T.; Goodfellow, I. J.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved Techniques for Training GANs. *CoRR*, abs/1606.03498.
- Saxena, D.; and Cao, J. 2020. Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions. *CoRR*, abs/2005.00065.
- Schlegl, T.; Seeböck, P.; Waldstein, S. M.; Schmidt-Erfurth, U.; and Langs, G. 2017. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *Information Processing in Medical Imaging*, 146–157.
- Stephenson, M.; and Renz, J. 2016a. Procedural generation of complex stable structures for angry birds levels. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8.
- Stephenson, M.; and Renz, J. 2016b. Procedural Generation of Levels for Angry Birds Style Physics Games. In *Proceedings of the Twelfth AAAI Conference on Artificial In-*

telligence and Interactive Digital Entertainment, AIIDE'16. AAAI Press. ISBN 978-1-57735-772-8.

Stephenson, M.; and Renz, J. 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.

Stephenson, M.; Renz, J.; Ge, X.; Ferreira, L.; Togelius, J.; and Zhang, P. 2019. The 2017 AIBIRDS Level Generation Competition. *IEEE Transactions on Games*, 11(3): 275–284.

Subramanian, S.; Rajeswar, S.; Dutil, F.; Pal, C.; and Courville, A. 2017. Adversarial Generation of Natural Language. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, 241–251. Vancouver, Canada: Association for Computational Linguistics.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2017. Procedural Content Generation via Machine Learning (PCGML). *CoRR*, abs/1702.00539.

Tanabe, T.; Fukuchi, K.; Sakuma, J.; and Akimoto, Y. 2021. Level Generation for Angry Birds with Sequential VAE and Latent Variable Evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21*, 1052–1060. New York, NY, USA: Association for Computing Machinery. ISBN 9781450383509.

Togelius, J.; Shaker, N.; Karakovskiy, S.; and Yannakakis, G. N. 2013. The Mario AI Championship 2009-2012. *AI Magazine*, 34(3): 89–92.

Torres-Reyes, N.; and Latifi, S. 2019. Audio Enhancement and Synthesis using Generative Adversarial Networks: A Survey. *International Journal of Computer Applications*, 182: 27–31.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A. M.; 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 (GECCO 2018)*. New York, NY, USA: ACM.

Xia, B.; Bai, Y.; Yin, J.; Li, Y.; and Xu, J. 2020. LogGAN: a Log-level Generative Adversarial Network for Anomaly Detection using Permutation Event Modeling. *Information Systems Frontiers*, 23(2): 285–298.

Xu, B.; Wang, N.; Chen, T.; and Li, M. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*, abs/1505.00853.