# Generative Models for Developing New Video Game Environments

1st Zaraar Malik
*Department of Artificial Intelligence*
*National University of Computer and Emerging Science*
Islamabad, Pakistan
i212705@nu.edu.pk

2nd Ahmed Kamal
*Department of Artificial Intelligence*
*National University of Computer and Emerging Science*
Islamabad, Pakistan
i210308@nu.edu.pk

3rd Raffay Khan
*Department of Artificial Intelligence*
*National University of Computer and Emerging Science*
Islamabad, Pakistan
i210335@nu.edu.pk

***Abstract*—Generative models have emerged as transformative tools in video game design, enabling the creation of immersive and diverse environments with minimal manual intervention. This paper explores the application of generative models, such as Variational Autoencoders (VAEs) [1], Generative Adversarial Networks (GANs) [2], and diffusion models, to synthesize novel and contextually rich game landscapes. These models facilitate the generation of procedurally varied terrains, dynamic textures, and interactive elements that adapt to gameplay, enhancing player engagement and reducing development overhead. Furthermore, we investigate the integration of domain-specific constraints, enabling the synthesis of environments aligned with artistic direction, gameplay mechanics, and narrative coherence. By leveraging large-scale training datasets and fine-tuning techniques, the generative approach demonstrates scalability across genres, from open-world explorations to arcade-style games. The study also addresses challenges such as balancing creative autonomy with developer control, optimizing computational resources, and ensuring visual fidelity. Ultimately, generative models represent a paradigm shift in game development, empowering creators to prototype and produce compelling environments more efficiently while fostering innovation in player experiences.**

***Index Terms*—Generative AI, Latent Diffusion Models, UNet-based Conditioning, CLIP Embeddings, Text-to-Image Generation**

## I. Introduction

The rapid evolution of video game technology has transformed the landscape of interactive entertainment, pushing the boundaries of creativity and immersion. As players increasingly seek novel experiences, the demand for diverse and expansive game environments has surged. Traditional methods of game design, often reliant on manual creation and static assets, struggle to keep pace with this demand, leading to a growing interest in innovative approaches that leverage artificial intelligence. Among these, generative models have emerged as a powerful tool for automating and enhancing the creation of video game environments.

Generative models, particularly those rooted in machine learning [3], offer the potential to revolutionize the way game worlds are conceived and constructed. By harnessing vast datasets of existing game environments, these models can learn intricate patterns and relationships, enabling them to generate new, unique landscapes that maintain coherence and aesthetic appeal. This capability not only streamlines the development process but also empowers designers to explore uncharted territories of creativity, allowing for the generation of environments that are both procedurally rich and contextually relevant. The implications of this technology extend beyond mere efficiency; they challenge the very nature of creativity in game design, prompting a reevaluation of the roles of human designers and AI collaborators.

In recent years, the gaming industry has witnessed a paradigm shift towards procedural generation, where algorithms create content dynamically rather than relying solely on pre-designed assets. This approach has been successfully implemented in various genres, from open-world adventures to roguelike games, where the unpredictability of generated environments enhances replayability and player engagement. However, the integration of generative models into game development is not without its challenges. Issues such as ensuring the quality and coherence of generated content, addressing potential biases in training data, and maintaining a balance between automation and artistic intent are critical considerations that developers must navigate.

This research paper delves into the various types of generative models—such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and procedural generation techniques—and their applications in the realm of video game environment design. We will examine case studies that illustrate the successful integration of these models into game development pipelines, highlighting their impact on gameplay, narrative, and player engagement. For instance, we will explore how GANs have [4] been utilized to create visually stunning landscapes that adapt to player actions, or how VAEs can generate intricate architectural designs that enhance the storytelling aspect of a game. These examples will serve to illustrate the transformative potential of generative models in crafting immersive worlds that resonate with players on multiple levels. Furthermore, we will address the challenges and ethical considerations associated with the use of generative models, including issues of originality, ownership, and the potential for bias in generated content. As the line between human creativity and machine-generated content blurs, ques-

Identify applicable funding agency here. If none, delete this.

tions arise about the authorship of generated environments and the implications for intellectual property rights.

## II. Variational Autoencoders (VAEs) Game Environment Generation

### A. Introduction

At its core, a VAE consists of two main components: the encoder and the decoder. The encoder maps input data ( x ) (e.g., images of game environments) into a lower-dimensional latent space ( z ), capturing the essential features of the data while discarding noise. The latent representation is typically modeled as a probability distribution, often assumed to be Gaussian:

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \tag{1}$$

where ( $\boldsymbol{\mu}$ ) and ( $\boldsymbol{\sigma}$ ) are the mean and standard deviation vectors produced by the encoder.

The decoder then reconstructs the original data from the latent space, enabling the generation of new content. The reconstruction is modeled as:

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \sigma^2\mathbf{I}) \tag{2}$$

### B. Training VAEs for Game Environments

To use VAEs for procedural content generation in game environments, the first step is to curate a dataset of existing game levels or environments. This dataset can include various elements such as terrain features, architectural styles, and environmental assets. The VAE is then trained on this dataset, learning to encode the essential characteristics of the environments into the latent space.

The training objective of a VAE is to maximize the evidence lower bound (ELBO), which can be expressed as:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = \mathbb{E}q\phi(\mathbf{z}|\mathbf{x})[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})|p(\mathbf{z})) \tag{3}$$

where:

1) ”L” is the ELBO,
2) ”theta” and ”phi” are the parameters of the decoder and encoder, respectively,
3) ”KL” is the Kullback-Leibler divergence,
4) p(z) is the prior distribution over the latent variables, typically a standard normal distribution N(0, I).

### C. Generating Diverse Game Environments

One of the most significant advantages of using VAEs for procedura [5] content generation is their ability to produce a wide variety of game environments. By sampling different points in the latent space, developers can generate unique levels that retain the stylistic and structural characteristics of the training data. This capability is particularly valuable in open-world games or sandbox environments, where players expect a rich and varied experience.

Moreover, VAEs allow for controlled generation by manipulating the latent variables. For example, developers can adjust specific dimensions of the latent space to create environments with desired features, such as increasing the amount of water or altering the terrain’s elevation. This level of control enables designers to tailor the generated content to fit specific gameplay mechanics or narrative themes.

## III. Understanding the GAN Architecture

### A. Introduction

A GAN consists of two neural networks: the generator ( G ) and the discriminator ( D ). The generator is responsible for creating synthetic data samples, while the discriminator evaluates the authenticity of these samples by distinguishing between real data from the training set and fake data produced by the generator.

The generator takes random noise ( z ) as input and produces a synthetic sample ( x-fake ):

$$\mathbf{x}_{\text{fake}} = G(\mathbf{z}) \tag{4}$$

The discriminator, on the other hand, takes both real samples ( xreal ) and fake samples ( xfake ) as input and outputs a probability ( D(x) ) that indicates whether the input is real or fake:

$$D(\mathbf{x}) = P(\text{real}|\mathbf{x}) \tag{5}$$

### B. Training GANs for Game Environments

The training of GANs involves a two-player minimax game between the generator and the discriminator. The objective is to optimize the following loss functions:

Discriminator Loss: The discriminator aims to maximize its ability to correctly classify real and fake samples:

$$\mathcal{L}D = -\mathbb{E}\mathbf{x}\text{real} \sim p\text{data}[\log D(\mathbf{x}\text{real})] - \mathbb{E}\mathbf{z} \sim p_z[\log(1 - D(G(\mathbf{z})))] \tag{6}$$

Generator Loss: The generator aims to minimize the discriminator’s ability to distinguish between real and fake samples:

$$\mathcal{L}G = -\mathbb{E}\mathbf{z} \sim p_z[\log D(G(\mathbf{z}))] \tag{7}$$

Here, ( p-data ) is the distribution of the real data, and ( p-z ) is the distribution of the random noise input to the generator.

### C. Generating Diverse Game Environments

One of the most significant advantages of using GANs for procedural content generation is their ability to produce high-quality and diverse game environments. By sampling different points from the noise distribution ( p-z ), developers can generate unique levels that maintain the stylistic and structural characteristics of the training data. This capability is particularly valuable in games that require a variety of environments, such as open-world or sandbox games.

GANs can also be conditioned on specific attributes to control the generation process. For instance, Conditional GANs (cGANs) can be used to generate environments based on certain conditions, such as terrain type or difficulty level. The generator in a cGAN takes both random noise and a condition ( c ) as input:

$$\mathbf{x}_{\text{fake}} = G(\mathbf{z}, c) \tag{8}$$

## IV. Diffusion Models for Video Game Generation

Diffusion models have emerged as a powerful tool for generating high-quality data by modeling the process of adding noise to data and then learning to reverse this process. This section outlines how diffusion models can be effectively utilized for procedural content generation (PCG) in video games, focusing on their architecture, training process, and practical applications.

### *A. Introduction*

Diffusion models operate by gradually adding noise to data until it becomes indistinguishable from random noise. The model then learns to reverse this process, effectively denoising the data to generate new samples. The key steps in this process include:

### *B. Latent Diffusion Process*

The latent diffusion model operates in a lower-dimensional feature space rather than directly on image pixels, significantly reducing computational overhead. Given an original image $x_0$, it is first encoded into a latent representation $z_0$ using an encoder $E$:

$$z_0 = E(x_0) \tag{9}$$

where $z_0$ is the latent vector representing the compressed version of $x_0$.

The forward diffusion process gradually adds Gaussian noise to $z_0$ over $T$ timesteps, resulting in a noisy latent vector $z_T$ at the final step. For each timestep $t$, noise is added according to a variance schedule defined by $\beta_t$:

$$q(z_t|z_{t-1}) = \mathcal{N}(z_t; \sqrt{1-\beta_t} \cdot z_{t-1}, \beta_t \cdot I) \tag{10}$$

where $\mathcal{N}$ represents a normal distribution, and $I$ is the identity matrix. By the final timestep $T$, $z_T$ approximates a Gaussian distribution, ensuring a smooth transition in the reverse denoising process.

### *C. Reverse Denoising Process*

The reverse process involves learning to progressively denoise $z_t$ from $t = T$ to $t = 0$, recovering a latent representation close to the original $z_0$. At each timestep $t$, a neural network $\epsilon_\theta$ predicts the noise component $\epsilon$ in $z_t$, conditioned on both the timestep $t$ and the text embedding. The denoising update for $z_{t-1}$ is given by:

$$z_{t-1} = \frac{1}{\sqrt{1-\beta_t}} \left(z_t - \beta_t \cdot \epsilon_\theta(z_t, t, c)\right) + \sigma_t \cdot z \tag{11}$$

where $c$ is the conditioning input from the text embedding, $\sigma_t$ is a scaling factor for noise, and $z \sim \mathcal{N}(0, I)$ represents Gaussian noise. This iterative process removes noise in each step, allowing the model to gradually reconstruct the latent representation.

### *D. Text Conditioning with CLIP Embeddings*

The text conditioning mechanism leverages CLIP embeddings to guide the generation process. Given a text prompt, the CLIP model generates a text embedding $c$ that captures the semantic content of the description:

$$c = \text{CLIP}_{\text{Text}}(\text{prompt}) \tag{12}$$

This embedding $c$ is then incorporated into the denoising function $\epsilon_\theta(z_t, t, c)$ to steer the model towards generating a latent representation aligned with the input text. By conditioning on $c$, the model learns to associate specific visual patterns with the text prompt, resulting in images that closely match the intended description.

### *E. UNet Architecture*

A UNet serves as the core of the denoising network $\epsilon_\theta$. The UNet architecture is well-suited for generative tasks due to its encoder-decoder structure, which captures both local and global features effectively. The UNet processes the noisy latent vector $z_t$ along with the text embedding $c$, using skip connections between the encoder and decoder paths to retain fine-grained details.

For each timestep $t$, the UNet takes $z_t$ and $c$ as inputs, and outputs an estimate of the noise $\epsilon$. The forward pass in the UNet can be expressed as:

$$\epsilon_\theta(z_t, t, c) = \text{UNet}(z_t, c, t) \tag{13}$$

where the UNet is parameterized by $\theta$ and conditioned on both $c$ and $t$. The skip connections in the UNet allow information from earlier layers (high-resolution features) to be directly passed to later layers, helping the model maintain spatial coherence during denoising.

### *F. Reconstruction of the Image*

After denoising through all timesteps, the final latent representation $z_0$ is obtained, which approximates the original latent vector. This denoised latent vector is then passed through the decoder $D$ to reconstruct the image:

$$\hat{x}_0 = D(z_0) \tag{14}$$

where $\hat{x}_0$ represents the generated image, and $D$ is the decoder part of the autoencoder trained alongside the encoder. The final output image $\hat{x}_0$ is expected to capture the semantic alignment specified by the text prompt, completing the text-to-image synthesis process.

## V. Data Set

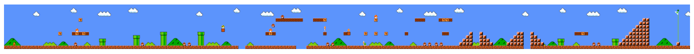
### *A. Introduction*

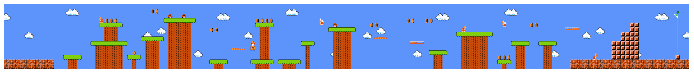
Generative Adversarial Networks (GANs) for image-to-image generation typically require large datasets of paired or unpaired images. Paired datasets consist of input-output image pairs that define a direct mapping, such as converting sketches to photorealistic images or grayscale images to colored ones. Unpaired datasets, on the other hand, enable training in scenarios where corresponding image pairs are unavailable, such as translating artistic styles or generating environments. These datasets need to be high-quality, diverse, and representative of the domain of interest to ensure that the GAN effectively learns the underlying patterns and textures. Common sources of such data include curated image repositories, game asset libraries, and public datasets tailored to the target application.

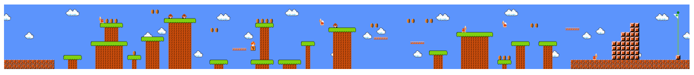
### B. Dataset Organization

To assemble a robust dataset for my work, I explored various sources on the web, focusing particularly on GitHub repositories that host open-source content. These repositories provided a wealth of material, ranging from terrain tilesets and object sprites to full-level layouts and design sketches. By utilizing repositories like The Video Game Level Corpus [6] and other community-driven collections, I was able to gather diverse data relevant to video game environments. I further complemented this with custom-extracted assets from gameplay videos and tools such as Lunar Magic, which allowed me to extract level data directly from ROM files. This extensive data collection process ensured that my dataset encompassed a wide variety of styles and themes.

### C. Some Sample images

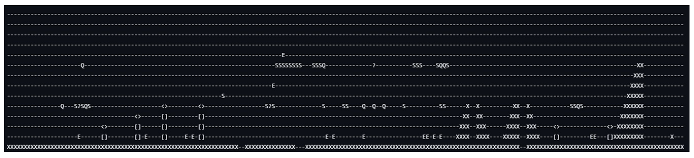


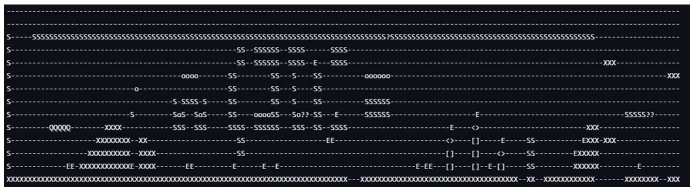


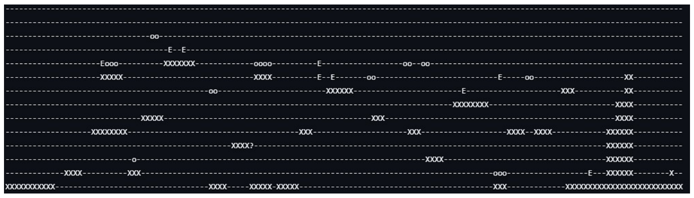


### D. Data Preprocessing

To prepare the collected data for training, I encoded the variables in a structured manner. Each tile or game object was represented using distinct symbols and encoded with their ASCII values to create a compact, machine-readable format. For instance, ground tiles were encoded with "G," enemies with "E," and air spaces with "-." These symbols were then converted into numeric ASCII representations, providing a standardized format for input into the GAN model. This encoding method not only simplified the data preprocessing process but also ensured that the model could effectively learn the spatial relationships and patterns within the dataset.







### E. Sample encodings

- **X**: Encoded as Solid, Ground
- **S**: Encoded as Solid, Breakable
- **-**: Encoded as Passable, Empty
- **?**: Encoded as Solid, Question Block, Full Question Block
- **Q**: Encoded as Solid, Question Block, Empty Question Block
- **E**: Encoded as Enemy, Damaging, Hazard, Moving
- **¡**: Encoded as Solid, Top-Left Pipe, Pipe
- **¿**: Encoded as Solid, Top-Right Pipe, Pipe
- **[**: Encoded as Solid, Left Pipe, Pipe
- **]**: Encoded as Solid, Right Pipe, Pipe
- **o**: Encoded as Coin, Collectable, Passable
- **B**: Encoded as Cannon Top, Cannon, Solid, Hazard
- **b**: Encoded as Cannon Bottom, Cannon, Solid

## VI. Generative Adversarial Networks

The architecture of a Generative Adversarial Network (GAN) (Goodfellow et al. 2014) can be understood as an adversarial game between a generator (G), which maps a latent random noise vector to a generated sample, and a discriminator (D), which classifies generated samples as real or fake. These adversaries are trained at the same time, striving towards reaching a state where the discriminator maximizes its ability to classify correctly and the generator learns to create new samples that are good enough to be classified as genuine. GANs became popular in recent years due to their impressive results in tasks such as image generation. However, training GANs is not a trivial procedure: the training process is often unstable, where the generator produces unrealistic samples, or the discriminator is no more accurate than a coin toss. For these reasons, many extensions have been proposed to improve the training process and the quality of the results. For example, Mirza and Osindero (2014) feed a vector y to train G and D conditioned to generate descriptive tags which are not part of training labels. In addition, Bellemare et al. (2013) proposed a new training methodology to grow both the generator and discriminator architecture complexity progressively, reaching a higher-quality on the CELEBA dataset. More recently, high-quality results have been reported using attention mechanisms in deep learning (Vaswani et al. 2017). Attention mechanisms are a very simple idea that identifies the most relevant variables dynamically in more complex deep neural network architecture such as, convolutional neural network (CNN). Recently, Zhang et al. (2018) combined attention mechanism and GANs to generate and discriminate high-resolution details as a function of only spatially local points in lower-resolution feature maps.

## VII. Proposed Methodology

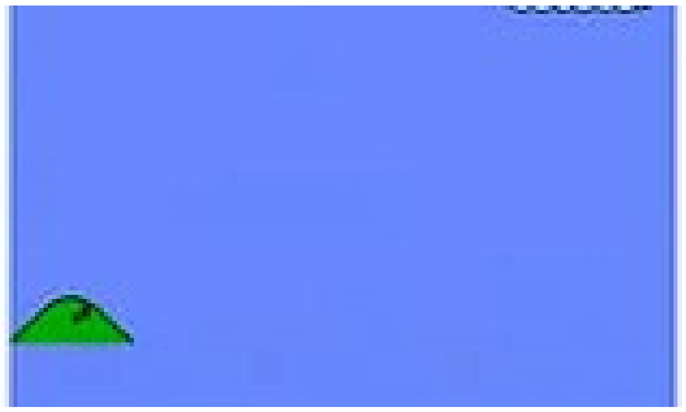
Training a Generative Adversarial Network (GAN) for image-to-image generation involves a well-structured process encompassing dataset preparation, model design, and iterative training. The first step is to gather and preprocess the data. For image-to-image generation tasks, paired or unpaired datasets are needed. Paired datasets consist of input-output image pairs that define a direct mapping, such as converting grayscale images to color or sketches to photorealistic representations. Unpaired datasets, used for tasks like style transfer, involve two domains without explicit mappings. Images must be resized to a uniform size, normalized to a specific range (e.g., [-1, 1] or [0, 1]), and augmented with transformations like flipping, cropping, or rotation to enhance diversity and improve generalization.

The second step involves designing the GAN architecture. A typical GAN comprises a generator and a discriminator. The generator takes input data and produces images, aiming to mimic the style and features of the target domain. For image-to-image tasks, architectures like encoder-decoder models or U-Net with skip connections are commonly employed, as they help preserve spatial features. The discriminator, on the other hand, evaluates the authenticity of generated images, distinguishing them from real ones. Patch-based discriminators like PatchGAN are often used, as they focus on smaller image regions, ensuring attention to fine-grained details. Loss functions play a vital role in training; adversarial loss ensures the generator learns to fool the discriminator, while a content loss (such as L1 or L2 loss) ensures the generated images closely resemble the target images.

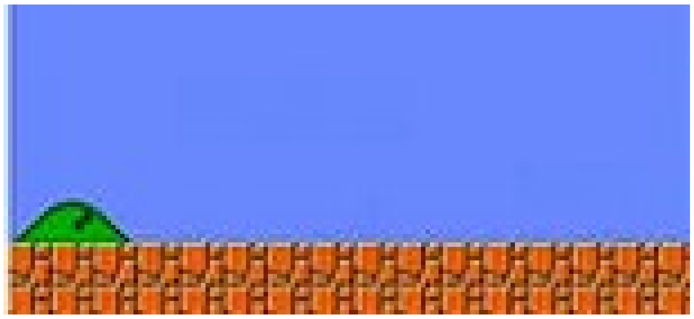
Once the architecture is in place, the training process begins. Training a GAN is an iterative process where the generator and discriminator are trained alternately. The generator generates an image, and the discriminator evaluates it. Feedback from the discriminator is used to refine the generator's ability to create realistic outputs. The process involves balancing the performance of the two networks, as an overly strong discriminator can prevent the generator from learning, while a weak discriminator fails to provide meaningful feedback. Over multiple epochs, the model's performance is evaluated visually by generating sample images and quantitatively using metrics like Structural Similarity Index (SSIM) or Peak Signal-to-Noise Ratio (PSNR). Once the training achieves satisfactory results, the trained generator can be saved and deployed to perform the image-to-image translation task for real-world applications. Fine-tuning the model through hyperparameter adjustments or additional training iterations can further enhance its performance.

Sprites that stand on the ground or other objects, sometimes grouped in twos and threes.

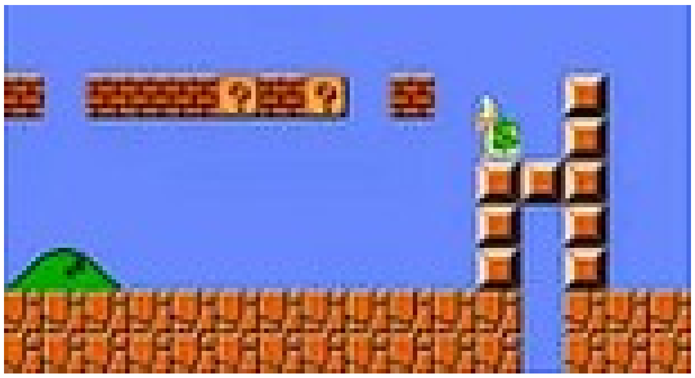
### A. *Epoch 1000*



### B. *Epoch 3000*



### C. *Epoch 5000*



We're also going to use some common tricks like label swapping and exponential averaging to improve our results. All of this is implemented in Python using TensorFlow as the backend and Keras to build the model.At the start of the training process the network doesn't know anything about the dataset and it essentially outputs random noise. However, as the training progresses the network eventually starts to generate levels that are more "Mario like".

## VIII. Conclusion

In conclusion, the research conducted on various generative models, including Variational Autoencoders (VAEs), GANs, and Diffusion Models, has led to valuable insights into their applicability for game level generation. Throughout the research, I explored different architectures and their potential for synthesizing novel and coherent game environments, with a particular focus on the GAN-based approach for this task. The initial exploration of VAEs and Diffusion Models provided useful theoretical grounding, as these models are known for their ability to learn complex latent representations and generate data in a controlled and meaningful way. While VAEs offer strong benefits in terms of encoding and reconstructing data efficiently, and Diffusion Models provide superior quality in image generation through iterative refinement, it was the GAN model that proved to be the most effective for the specific task of game level generation.

The choice to implement and train a GAN for this task was driven by the desire to create realistic and diverse game levels that can mimic the inherent patterns found in real-world game design. GANs, with their adversarial nature, allowed the model to generate new levels that were both highly varied and capable of passing the discriminator's evaluation as "realistic," based on the training data. The approach I followed involved a systematic collection of data from various open-source game repositories, encoding the data in a format that allowed the generator to understand and learn from the structure of game levels. The GAN model's architecture, designed with a generator and discriminator, was trained iteratively to refine the generator's ability to produce realistic game levels while maintaining diversity in the generated outputs.

The results of the GAN-based approach were promising. The model generated game levels that exhibited coherent spatial layouts and consistent use of objects, which are essential qualities for any playable game level. Though the results were not perfect, and some generated levels exhibited inconsistencies or errors typical of GAN-based generation, the overall quality was reasonable and demonstrated the model's potential for creating playable and diverse levels. The generator showed promise in capturing the fundamental characteristics of game levels, such as object placement, terrain types, and level flow, which are critical to maintaining the gaming experience. The discriminator effectively guided the training, improving the generator's ability to refine its output over time.

This research underscores the potential of GANs in the field of procedural content generation (PCG) for games, where the goal is to generate new, engaging environments for players. While GANs are not without their challenges—such as the need for large and diverse datasets and the tendency of the generator to produce artifacts or unrealistic elements—the results from this project show that with proper tuning and data preparation, GANs can be a powerful tool for generating game levels. The work done here also opens doors for future exploration of hybrid models that combine the strengths of VAEs, Diffusion Models, and GANs, which could further improve the quality and diversity of the generated content.

In summary, this research demonstrates that GANs, with their ability to generate high-fidelity and realistic outputs, are well-suited for game level generation. The experience gained from experimenting with VAEs and Diffusion Models enhanced the understanding of generative processes and informed the decision to focus on GANs for this particular task. Moving forward, further optimization and refinements to the GAN model, including improved loss functions, more sophisticated discriminator architectures, and larger training datasets, could yield even better results. The approach presented here has the potential to contribute significantly to the field of procedural content generation, providing game developers with tools to automatically generate diverse, engaging, and playable game levels, ultimately enhancing the gaming experience for players.

## References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, Springer, 2015.

[3] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi, "Image super-resolution via iterative refinement," *arXiv preprint arXiv:2104.07636*, 2022.

[4] E. Whiteway, "using-a-generative-adversarial-network-to-author-playable-super-mario-bros-levels," 2021.

[5] T. Shu, J. Liu, and G. N. Yannakakis, "Experience-driven PCG via reinforcement learning: A super mario bros study," *CoRR*, vol. abs/2106.15877, 2021.

[6] A. J. Summerville, S. Snodgrass, M. Mateas, and S. O. n'on Villar, "The vglc: The video game level corpus," *Proceedings of the 7th Workshop on Procedural Content Generation*, 2016.