A NEW APPROACH TO INCREASE VARIABILITY AND PLAYABILITY VIA GAME BLENDING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

ÖMER FARUK KARAKAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JULY 2023

A New Approach to Increase Variability and Playability via Game Blending

submitted by **ÖMER FARUK KARAKAYA** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems Department, Middle East Technical University** by,

Date: 24.07.2023

Name, Surname: Ömer Faruk Karakaya

Signature :

ABSTRACT

A NEW APPROACH TO INCREASE VARIABILITY AND PLAYABILITY VIA GAME BLENDING

Karakaya, Ömer Faruk
M.S., Department of Information Systems
Supervisor: Assoc. Prof. Dr. Elif Sürer
Co-Supervisor: Assoc. Prof. Dr. Aysu Betin Can

July 2023, 79 pages

In this thesis, the objective is blending game levels in a way that retains and ensures playability, using advanced procedural content generation techniques. The motivation stems from the aim to increase the diversity and replayability of games and concurrently reduce the extensive effort required for manual design. A novel approach to game blending is examined, unifying The General Video Game AI (GVGAI) game descriptions with level representations, resulting in a solution with broad applicability across a multitude of games. To address the research question effectively, the use of Variational Autoencoder Generative Adversarial Networks (VAEGANs) is introduced, a pioneering hybrid model which combines the unique strengths of Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to facilitate superior level blending and generation. A focus has been placed on enhancing playability, incorporating A* algorithms and Reinforcement Learning (RL) agents to optimize latent vectors in generative networks via the Covariance Matrix Adaptation Evolution Strategy. Through several experimentations, the performance of GANs, VAEs, and VAEGANs in blending game levels was assessed, discovering their capacity to create novel, diverse, and playable mixed levels. VAEGANs emerged as the superior model, producing distinct, intricate, and varied levels surpassing the capabilities of VAEs and GANs. This enhanced performance owes its success to the combination of the encoder-decoder architecture and adversarial training, allowing VAEGANs to utilize the strengths of both models while avoiding some of their individual drawbacks.

Keywords: Level Blending, Level Generation, Procedural Content Generation, Generative Adversarial Networks, Variational Autoencoders

ÖZ

OYUN HARMANLAMA YOLUYLA ÇEŞİTLİLİĞİ VE OYNANABİLİRLİĞİ GELİŞTİRMEK İÇİN YENİ BİR YAKLAŞIM

Karakaya, Ömer Faruk
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Doç. Dr. Elif Sürer
Ortak Tez Yöneticisi: Doç. Dr. Aysu Betin Can

Temmuz 2023, 79 sayfa

Bu tez kapsamında gelişmiş Prosedürel İçerik Oluşturma (PCG) teknikleri kullanılarak oyun düzeylerinin oynanabilirliğini sağlayacak şekilde farklı oyunlar birleştirilmektir. Tezin amacı, oyunların çeşitliliğini ve tekrar oynanabilirliğini artırmak ve aynı zamanda manuel tasarım için gereken kapsamlı çabayı azaltmaktır. Bu çalışmada, Genel Oyun Yapay Zeka (GVGAI) oyun açıklamalarını ve oyun seviyesi tanımlarını bir araya getirerek oyunları birleştirmeye yönelik yeni bir yaklaşım incelenmiştir ve çok sayıda oyunda geniş uygulanabilirliğe sahip bir çözüm ortaya çıktığı gözlenmiştir. Araştırma sorusunu etkili bir şekilde ele almak için, öncü bir hibrit model olan Variational Autoencoder Generative Adversarial Networks (VAEGANs) kullanılmıştır. Bu hibrit model, içerik oluşturmayı kolaylaştırmak için Varyasyonel Otomatik Kodlayıcıların (VAEs) ve Çekişmeli Üretici Ağların (GANs) özelliklerini bir araya getirmektedir. Kovaryans Matrisi Uyarlama Evrimi Stratejisi (CMA-ES) aracılığıyla üretken ağlardaki gizli vektörleri optimize etmek için A* algoritmalarını ve Pekiştirmeli Öğrenme (RL) ajanlarını dahil ederek oynanabilirliği sağlamaya odaklanılmıştır. Çeşitli deneyler yoluyla, GAN, VAE ve VAEGAN yöntemlerinin oyun seviyelerini harmanlamadaki performansı değerlendirilmiştir ve yeni, çeşitli ve oynanabilir karışık seviyeler yaratma kapasiteleri incelenmiştir. VAEGAN, VAE ve GAN yöntemlerinin yeteneklerini aşan farklı, karmaşık ve çeşitli düzeyler üreten üstün model olarak ortaya çıkmıştır. Bu geliştirilmiş performans, başarısını VAEGAN yönteminin her iki modelin de güçlü yönlerinden yararlanırken modellerin bazı dezavantajlarından kaçınmasına olanak tanıyan kodlayıcı-kod çözücü mimarisi ile çekişmeli eğitimin birleşimine borçludur.

Anahtar Kelimeler: Oyun Harmanlama, Oyun Seviyesi Üretimi, Prosedürel İçerik Oluşturma, Çekişmeli Üretici Ağlar, Otomatik Kodlayıcılar

To my family and friends

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Assoc. Prof. Elif Sürer and my co-supervisor Assoc. Prof. Aysu Betin Can for their guidance, encouragement, and endless patience throughout the completion of this thesis. I have learned a lot during our discussions.

I would like to express my deepest gratitude to Prof. Banu Günel Kılıç, Assoc. Prof. Ebru Aydın Göl, and Assoc. Prof. Burkay Genç, for being a part of this thesis jury and giving invaluable feedback.

I wish to express my heartfelt gratitude to my friends. Their unwavering support and reliability have been invaluable to me.

The journey of completing this thesis would not have been possible without the steadfast backing of my family. I am deeply appreciative of their continual encouragement, patience, understanding, and support throughout the entire process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ESM	Experience Sampling Method
GLMM	Generalized Linear Mixed Model
PCG	Procedural Content Generation
RNNs	Recurrent Neural Networks
LSTM	Long Short-Term Memory
GAN	Generative Adversarial Network
WGAN	Wasserstein Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
CNN	Convolutional Neural Network
VAE	Variational Autoencoder
KL	Kullback-Leibler
VAEGAN	Variational Autoencoder Generative Adversarial Networks
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
GVGAI	General Video Game Artificial Intelligence
AI	Artificial Intelligence
VGDL	Video Game Description Language
RL	Reinforcement Learning
PPO	Proximal Policy Optimization
A2C	Advantage Actor-Critic
VGLC	Video Game Level Corpus

CHAPTER 1

INTRODUCTION

Procedural content generation (PCG) is a technique that uses algorithms to create levels, characters, landscapes, and other game content without relying on manual design. This method helps increase the amount and variety of game content, making the gameplay more diverse and enjoyable. PCG includes different approaches, such as random and rule-based generation, search-based techniques, and evolutionary and machine learning-based generation.

Traditional techniques rely on pseudo-random number generators and constructive methods for their simplicity and quick generation. They often used to create dungeons and labyrinths. These deterministic methods create the same content with the same seed values. This trait was initially used for data compression [1, 2]. In addition, fractal and noise techniques are used to create digital versions of natural events, such as the shaping of mountains or the texture of rocks and plants, to give the content a more natural feel. They are frequently used in video games that showcase nature. Apart from that, grammars have been implemented successfully in various fields, including video games, architecture, filmmaking, generating cities, and creating levels for action games [3, 4, 5].

In PCG academic research, search-based methods evaluate and rate generated content instead of just accepting or rejecting it. This evaluation involves using an optimization or search algorithm, such as an evolutionary algorithm, until the content meets specific criteria or achieves a high enough score, verified by another algorithm [6]. In addition, evolutionary generation uses techniques like genetic algorithms to create more exciting and complex content, such as levels or items [7]. The method has three components: representation of space, evaluation function, and search algorithm. The evolutionary creation approach has been used to create game space, systems, levels, and game rules [8, 9, 10].

The process of generating content using machine learning algorithms, such as deep learning, is known as machine learning-based generation. Machine Learning methods involve using techniques like Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM), Autoencoders, Generative Adversarial Networks (GANs), Convolutional Neural Networks (CNNs), and Markov Models to create levels or items that imitate those made by human designers [11]. Procedural generation techniques, such as LSTMs and Autoencoders, have become increasingly popular for tasks such as image classification and board game playing. In procedural content generation for games, these techniques are commonly used to generate game levels, which are then divided into sequences, grids, and graphs based on the game type. Super Mario Bros is an example of a game for which LSTMs and Autoencoders have been successfully used for procedural generation, and Starcraft maps were created using CNNs [12, 13, 14].

Game blending is a PCG technique that mixes elements from various games or genres to create fresh and unique content. Being able to blend games generates a vast array of options that are engaging to players. Game blending techniques include rule-based blending, genetic algorithms, and blending with machine learning. For instance, Frolda is a rule-based game blending system that blends game specifications from Frogger and Zelda to create a hybrid gaming experience [15]. By combining elements from different levels, genetic algorithms have been used for level blending to produce new and unique Mario levels [16]. Sarkar and Cooper used machine learning algorithms, specifically LSTMs and later VAEs, to blend different games. They blended levels from a variety of platformer games [17, 18]. Previous research showed that it is possible to create blended levels by using models trained on data from different games. However, these studies did not include actual gameplay environments for blended games.

Our research focuses on improving the playability of blended games. First, we propose a structural methodology for blending the General Video Game AI (GVGAI) game descriptions. Then, we employ GANs, VAEs, and their hybrid version–Variational Autoencoder Generative Adversarial Networks (VAEGANs)–for the procedural generation of blended game levels. Furthermore, we use the A* pathfinding algorithm and Reinforcement Learning (RL) agents in the heuristic score function of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to evolve the latent space of the GANs, VAEs, and VAEGAN models to optimize the results of the generator networks for playability.

For the experimental validation of our methods, we have selected the GVGAI environment [19]. We conducted experiments on level blending using four GVGAI sample games, namely Pacman, Dungeon, Zelda, and Sokoban. Our findings showed that our methods were effective in producing positive results. Furthermore, VAEGAN shows better performance by utilizing the advantages of both VAE and GAN networks, indicating its potential for creating hybrid games through procedural generation.

1.1 Research Questions

In this thesis, we ask the following research questions regarding game blending:

- Can we provide a systematic method for blending game descriptions?
- Which methodology is more suitable for blending game levels?
- Can we guarantee the playability of blended games?

1.2 Contributions of the Study

We list the contributions of this thesis as follows:

- A systematic method for blending game descriptions.
- Comparison of different generative networks GAN, VAE, and VAEGAN for game level blending.
- A method to provide playability in the results of generative networks.
- A strategy to apply A* search in conjunction with CMA-ES to optimize latent vectors of generative networks.

Additionally, we submitted a manuscript based on this thesis to a SCI-Expanded journal.

1.3 Organization of the Thesis

Chapter 2 presents fundamental concepts and methods used in this research. Chapter 3 surveys related research. Chapter 4 delineates our research methodology, explicating aspects such as level representation, blending game descriptions, training processes, generative models, and latent variable optimization of models. Next, Chapter 5 presents our experimental design and findings, segmented into the analysis of novelty, tile density, and diversity in generated levels. The ensuing discussion in Chapter 6 interprets our findings. The thesis wraps up in Chapter 7, summarizing our key insights and emphasizing the implications of our research in the broader context of game blending and procedural content generation.

CHAPTER 2

PRELIMINARIES

This chapter presents the preliminary material in the following sections: Generative Adversarial Networks (Section 2.1), Variational Autoencoders (Section 2.2), Variational Autoencoder Generative Adversarial Networks (Section 2.3), Covariance Matrix Adaptation Evolutionary Strategy (Section 2.4), General Video Game AI (Section 2.5), and Reinforcement Learning (Section 2.6). Each section provides a concise explanation of the important concepts associated with each topic and the mathematical models that support them.

2.1 Generative Adversarial Networks

Goodfellow and colleagues introduced a revolutionary innovation in the field of machine learning known as Generative Adversarial Networks (GANs) [20]. The framework has two connected parts. The first is a generator (G) that produces samples from random noise vectors. The other part is a discriminator (D), which evaluates whether the sample is real and gives a rating between 0 and 1. Generator and discriminator networks engage in a dynamic interplay that resembles an adversarial game between two players. The generator aims to create realistic samples, while the discriminator aims to become skilled at identifying imitations.

In this competitive process, the generator is trained to create samples that are so convincing that the discriminator cannot tell the difference between them and real ones. As the training progresses, the generator gets better and eventually becomes skilled at creating realistic samples, while the discriminator cannot accurately differentiate between real and generated samples.

Arjovsky, Chintala, and Bottou built upon this work with the introduction of the Wasserstein GAN (WGAN) [21]. By employing the Wasserstein distance, a measure of the cost of transporting mass, they solved the problem of unstable gradients experienced in traditional GANs, leading to more stable training and improved sample quality.

The generator and discriminator in a WGAN are expressed as [21]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[D(G(\mathbf{z}))] \tag{1}$$

This equation can be dissected into two parts. The first term, $\mathbb{E}\mathbf{x} \sim p\text{data}(\mathbf{x})[D(\mathbf{x})]$, is the expectation of the discriminator outputs over all real data. The second term, $\mathbb{E}\mathbf{z} \sim p\mathbf{z}(\mathbf{z})[D(G(\mathbf{z}))]$, is the expectation of the discriminator outputs over all generated data. The equation represents the struggle between the generator and the discriminator: the generator tries to minimize this function while the discriminator tries to maximize it.

The scope of GANs expands to include the Deep Convolutional Generative Adversarial Network (DCGAN) [22]. DCGANs are a significant evolution in the architecture of GANs that incorporate Convolutional Neural Networks (CNNs). This integration facilitates more efficient learning of spatial hierarchies of features, yielding samples of superior resolution and detail.

However, it is essential to note a significant challenge within the GAN paradigm known as Mode Collapse [23]. This issue emerges when the generator begins to generate less diverse outputs, often converging towards similar or identical results despite different inputs. Despite the possibility that the generator could deceive the discriminator, it is incapable of learning to represent the complex real-world data distribution and becomes trapped in a narrow space with little variation.

2.2 Variational Autoencoders

In 2013, Kingma and Welling created a generative model called Variational Autoencoders that uses deep learning concepts [24]. VAEs are unique because they have two parts: an encoder network that compresses data into a smaller space and a decoder network that can recreate the original data from that compressed space. In addition, they differ from other autoencoders because they can generate new samples using probabilities.

VAEs optimize the evidence lower bound, or variational lower bound, during the training phase. This limit includes the reconstruction loss and the Kullback-Leibler (KL) divergence. The reconstruction loss evaluates how well the reconstructed data matches the original input, while the KL divergence imposes a Gaussian prior distribution on the latent space. This setup helps with interpolation and sampling.

The loss function of VAEs is typically expressed in the following mathematical format [24]:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{2}$$

In this formulation, θ and ϕ represent the parameters of the decoder and encoder networks. $\mathbf{x}$ denotes the data, and $\mathbf{z}$ is the associated latent variable. The first term represents the reconstruction loss, while the second term represents the KL divergence between the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the prior distribution $p(\mathbf{z})$.

2.3 Variational Autoencoder Generative Adversarial Networks

In 2019, Yu et al. [25] introduced the concept of Variational Autoencoder Generative Adversarial Networks (VAEGANs), which combines the strengths of Variational Autoencoders and

Generative Adversarial Networks and addresses the limitations of both models. VAEs are unsupervised learning models known for their ability to encode data into a compact latent space and generate new samples by decoding them. However, they may produce blurry samples due to pixel-wise reconstruction loss. On the other hand, GANs are good at creating sharp and realistic samples but can face issues like training instability and mode collapse.

The VAEGAN framework combines the encoder-decoder structure of VAEs with the adversarial training mechanism of GANs to address these issues. The VAEGAN model has three primary parts: a generator G, a discriminator D, and an additional encoder E. The encoder E is responsible for converting the input data into the latent space, and the generator G transforms the latent space representations into new samples. The role of discriminator D is to differentiate between real and generated samples, similar to its function in GANs.

The VAEGAN loss function incorporates terms from both the VAE and GAN models but also includes additional components. In a typical Variational Autoencoder (VAE), the loss function is composed of two parts: the reconstruction loss and the Kullback-Leibler (KL) divergence.

The GAN part of the VAEGAN introduces an adversarial loss which is used to train the discriminator to distinguish between real and generated samples, and the generator to fool the discriminator.

In addition to these terms, the VAEGAN introduces a contractive penalty to the classical reconstruction cost function. This additional term helps to yield robust features and improve the generalization performance of the model. The contractive penalty is typically implemented by adding a term to the loss function that penalizes large derivatives of the encoder's outputs with respect to its inputs.

The VAEGAN loss function can thus be represented in the following way:

$$\mathcal{L}_{\text{VAEGAN}}(\theta,\phi,\psi;\mathbf{x}) = \mathcal{L}_{\text{rec}}(\theta,\phi;\mathbf{x}) + \lambda_1\mathcal{L}_{\text{KL}}(\phi;\mathbf{x}) + \lambda_2\mathcal{L}_{\text{adv}}(\theta,\psi;\mathbf{x}) + \lambda_3\mathcal{L}_{\text{con}}(\phi;\mathbf{x}) \quad (3)$$

In this equation, the VAEGAN loss function $\mathcal{L}_{\text{VAEGAN}}$ represents the total loss for the VAEGAN model, with parameters θ, ϕ, and ψ for the generator, encoder, and discriminator, respectively. The input data is denoted by $\mathbf{x}$.

The reconstruction loss $\mathcal{L}_{\text{rec}}$ measures the difference between the input data and the output of the decoder, encouraging the model to produce an output that closely resembles the original input data.

The Kullback-Leibler divergence term $\mathcal{L}_{\text{KL}}$ pushes the latent space to conform to a specific distribution, typically a standard normal distribution. This term acts as a regularization measure, preventing the model from encoding excessive information in the latent variables and promoting the learning of more generalized data representations.

The adversarial loss $\mathcal{L}_{\text{adv}}$ is used in the GAN portion of the VAEGAN for training the discriminator and the generator. The generator's goal is to generate samples that the discriminator cannot distinguish from real data, while the discriminator aims to correctly classify real and generated samples.

The contractive penalty term $\mathcal{L}_{\text{con}}$ encourages the model to produce robust features by penalizing large changes in the encoder's outputs relative to its inputs, thus aiding in the improvement of the model's generalization performance.

Lastly, the terms λ_1, λ_2, and λ_3 are the weights assigned to the KL divergence, adversarial loss, and contractive penalty terms respectively. These weights control the relative contribution of each component to the total loss and can be tuned based on the specific task at hand and the nature of the input data.

2.4 Covariance Matrix Adaptation Evolutionary Strategy

In 2003, Hansen and others introduced the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [26]. This evolutionary algorithm utilizes natural selection principles to solve complex optimization problems. The primary concept of CMA-ES can be encapsulated by the following three equations in which m represents the mean of the search distribution, C represents the covariance matrix, and p_σ and p_c represent the evolution paths:

$$m_{t+1} = m_t + c_\mu \sum_{i=1}^{\mu} w_i (x_{i:t} - m_t) \tag{4}$$

$$C_{t+1} = (1 - c_1 - c_\mu) C_t + c_1 (p_{c,t} p_{c,t}^T + (1 - h_t) C_t) \quad + c_\mu \sum_{i=1}^{\mu} w_i (x_{i:t} - m_t)(x_{i:t} - m_t)^T \tag{5}$$

$$p_{\sigma,t+1} = (1 - c_\sigma) p_{\sigma,t} + \sqrt{c_\sigma (2 - c_\sigma) \mu_{\text{eff}}} \frac{m_{t+1} - m_t}{\sigma_t} \tag{6}$$

The first equation updates the mean m for the next generation. It is adjusted based on the weighted sum of the deviations of the better half of the current generation from the current mean.

The second equation updates the covariance matrix C, which controls the shape of the multivariate normal distribution used to generate the offspring. The covariance matrix is updated based on the information from the evolution paths and the successful candidate solutions.

The final equation updates the evolution path p_σ, which is used in the adaptation of the step size. This equation combines the old evolution path with the current step made by the mean on the normalized fitness landscape.

These mathematical representations highlight the adaptable nature of CMA-ES, which allows it to dynamically shape the search distribution across generations and optimize solution search in complex spaces.

2.5 General Video Game AI

The General Video Game AI (GVGAI) platform offers various tools and environments for researchers and developers to create and test artificial intelligence (AI) agents in various game genres [19]. The main element of GVGAI is the Video Game Description Language (VGDL), a specific language created to precisely describe the mechanics, rules, and interactions for various types of video games [27]. Furthermore, the GVGAI platform comes with various pre-made VGDL game descriptions and tools for creating new ones, making it a flexible research platform well-suited for the rapidly developing field of game AI research.

2.6 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning that involves training agents to make a series of decisions, with the goal of maximizing some notion of cumulative reward. This concept of learning by interacting with an environment was introduced by Sutton and Barto [28]. The fundamental dynamics of RL can be captured through the following equation in which s represents the current state, a the action taken, s' the next state, r the reward received, and π the policy:

$$(s, a) \xrightarrow{\pi} (s', r) \tag{7}$$

This equation is a simplification of the RL process, which involves the agent observing the current state s, taking an action a according to its policy π, transitioning to the next state s', and receiving a reward r.

One of the most critical aspects of RL is the reward hypothesis, which suggests that all goals can be framed as the maximization of the expected cumulative reward. This is formally represented as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8}$$

Here, G_t is the total accumulated reward, or the return, from time step t onward, R_{t+k+1} is the reward at time step $t + k + 1$, and γ is the discount factor, which determines the present value of future rewards.

Another essential part of RL is the value function $V^\pi(s)$, which measures how good a state s is under a policy π. This function is represented as:

$$V^\pi(s) = \mathbb{E}\big[G_t \big| S_t = s\big] \tag{9}$$

The value function calculates the expected return when starting from state s and following policy π thereafter. It provides a measure of the quality of a state given a particular policy.

CHAPTER 3

LITERATURE REVIEW

In this chapter, the literature on game blending and game level generation is presented.

3.1 Game Level Generation

In this chapter, we explore various methodologies and approaches to game level generation. Game level generation can be broadly classified into three categories: traditional methods, search-based methods, and machine learning methods.

3.1.1 Traditional Methods

Traditional methods in game level generation include pseudo-random number generators, constructive methods, generative grammars, fractals, noise, cellular automata, and others [29]. The use of pseudo-random number generators and constructive methods marked the initial introduction of PCG in commercial video games, especially in the dungeon and labyrinth generation. These methods are known for their simplicity, speed, and deterministic nature, leading to their widespread usage as a means of data compression [5]. Generation through fractals and noise has found extensive use in simulating natural processes such as mountains created by geological processes or textures [29]. The use of grammar to generate levels is another significant case of successful PCG implementation in video games. Commercial applications like SpeedTree, city generation, and platformer games level generation have benefited from grammar-based PCG [4, 3, 30].

3.1.2 Search-based Methods

Search-based methods have attracted considerable academic research interest. They generate content, evaluate it, and then score it. The methods continue generating content until another algorithm tries to score high enough or it meets some predefined criteria [2]. Search-based methods typically involve three components: the space representation, the evaluation function, and the search algorithm. These methods have proven successful in generating a wide range of content, including game design, game levels, and other complex content.

3.1.3 Machine Learning Methods

The incorporation of machine learning methodologies into PCG represents a significant development in recent years. Notable methodologies include Recurrent Neural Networks (RNNs), Autoencoders, Generative Adversarial Networks (GANs), and Markov Models [12, 13, 31, 32]. The deployment of these techniques in PCG has demonstrated proficiency across an expansive range of game genres, encompassing card games, Role-Playing Games (RPGs), Real-Time Strategy (RTS) games, and open-world games [33, 34, 35, 36].

However, procedural content generation via machine learning (PCGML) has distinct challenges. It is crucial to highlight the contrast in generating game content when compared to other generative tasks such as creating images, or text. One significant distinction is the scarcity of data. Compared to other domains, games often provide a limited volume of training data. This lack of training data can hinder the efficacy of deep learning models, which typically perform well when there is a substantial amount of training data.

The functional requirements of game content have an additional difficulty. Numerous aspects of game content, such as levels, actors, and game rules must adhere to certain functional requirements to ensure playability. For example, game levels need to be designed so that players can discover keys to the exits. In the absence of this, no matter how appealing they might be, the levels become unplayable. This functional aspect of game content makes it more comparable to the program code than to images [30].

Despite these challenges, promising solutions emerge from the combination of machine learning with search-based methodologies. Methods like Latent Variable Evolution provide the capability to examine the space learned by a trained model, rather than simply taking samples from it. This technique has proven effective in creating functional content, such as generating segments for Super Mario Bros levels [37, 38].

3.2 Game Blending in PCG

Game level blending is an exciting part of procedural content generation. Game level blending combines elements from different game levels to create novel games. Since its beginning, the field has received various academic contributions to enhance game diversity and reduce manual design efforts. These contributions have provided unique insights and methodologies that have helped shape the domain.

The early studies focused on the possibilities of combining game descriptions created in VGDL, as mentioned in the article of Gow and Corneli [15]. However, their manual approach lacks a definitive procedural method for generating new game descriptions. In addition, they highlighted the difficulty of fully automating this process, as integrating game descriptions frequently requires contextual knowledge of game elements and mechanics.

One significant development in game-level blending has been the successful integration of GANs into PCG. A case in point is the experiment led by Volz and his team [38] on Super Mario Bros, which involved training a generator on pre-existing level designs. They train the generator on pre-existing level designs. As a result, GANs produced unique levels similar to the original game in terms of aesthetics and gameplay. Similarly, using GANs, Giacomello and his colleagues created new Doom levels with unique layouts and architectural features to enhance the game's replay value [39]. In another innovative application, Gutierrez and Schrum [40] employed GANs in designing rooms for generative graph grammar dungeons in The Legend of Zelda. They created visually compelling and mechanically functional rooms by conditioning the generator on previous dungeon room designs. Furthermore, Schrum and his team [41] discovered a new way to design levels for video games using interactive evolution and GANs' latent space. They created a system that allows users to influence the generation of level designs and refine them based on their preferences. This approach allows for more personalized gaming experiences.

VAEs, on the other hand, have found broad applications within PCG. For instance, they were used to generate levels for Lode Runner [42]. Yang and their colleagues [43] utilized Gaussian Mixture Variational Autoencoders to develop a system that can cluster and create game levels. Sarkar and his team [44] created a customized gameplay experience using VAEs to regulate the level blending of multiple games in a complex application. However, all those studies focused on artificially generating game levels of the corresponding games; they do not discuss blending game levels across different games.

Advanced methodologies like Conditional Variational Autoencoders (CVAEs) have been introduced in game level blending, which helped blend games from different genres. As a result, evaluating playability within blended levels has become a new challenge [44]. Also, using probabilistic models to learn from gameplay videos was a significant development in creating distinctive types of game levels designed to mimic how humans design them [16]. However, these studies did not focus on assuring the playability of blended games. Later, more sophisticated tools were introduced, such as using a variational autoencoder and a random forest classifier. These tools have helped create logical platformer levels and set the foundation for controlling blend proportions [18]. Recent progress in VAE-based level generation has included incorporating path information that has expanded blending techniques and opened up new possibilities for future research [45]. Furthermore, recent research has combined VAEs with the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm for creating varied and playable levels [46]. However, while those studies have shown that they are able to control outputs of VAE, which is an important step forward in generating playable games, they do not provide an alternative way to blend other game elements such as game objects, the interaction between game objects, and termination rules.

3.3 Reinforcement Learning in Game Environments

Reinforcement learning (RL) in game environments is an interesting area of study in artificial intelligence. Recent studies of RL in video games focus on the training of generic agents that can learn to perform well in multiple games. Various academic studies have contributed to our knowledge and methods since the very beginning of this field, which has shaped the development of this area of study.

The first studies focused on RL agents that play one game well. Mnih et al.'s classic work on DeepMind's DQN, which mastered Atari games, showed this early focus. These strategies were successful but too game-specific to transfer abilities to different environments [47].

Transfer learning has improved multi-game RL, and Sutton and Barto's Policy Gradient techniques established the framework [28]. Their original technique could not bridge the gap between distinct games, the high variance in their environments and rules. However, their work showed the necessity for algorithms that can understand and adapt to different game principles, enabling future advances.

In addition, OpenAI's Proximal Policy Optimization (PPO) algorithm was a big step forward for RL [48]. After being trained in many different jobs, the algorithm showed a remarkable capacity to learn how to play multiple games with just one algorithm. However, it is important to remember that it does not work the same way in all games and often needs to be tweaked to work best in each one.

The next thing DeepMind made, the Agent57, was a big step toward a more flexible AI that could play games [49]. It showed that it could beat people at 57 different Atari games, which is a big step forward in terms of performance and adaptability.

Within the GVGAI environment, Torrado et al. [50] integrated the GVGAI framework with OpenAI's gym in 2018, enabling training with deep RL agents from the library. In their work, the authors tested two versions of Deep Q Network (DQN) and an Advantage Actor Critic (A2C) agent on the first level of a set of eight games, without evaluating them on other levels. On a similar note, Justesen et al. [51] implemented A2C in a training environment that included the procedural generation of levels, where the agent was provided with levels of increasing difficulty based on its learning progress, showcasing another path toward generalization across games. In this thesis, we used a training environment developed by Justesen et al. to train RL agents that can play both games that are subject to be blended.

CHAPTER 4

METHODOLOGY

Blending game levels have been the subject of numerous studies; however, many of these methodologies fall short of providing a comprehensive framework encompassing all game elements, occasionally leading to blended levels that fall short of optimal playability in a gameplay context. This thesis attempts to surmount this challenge by proposing an inclusive method for blending VGDL game definitions and game-level representations. We follow a three-tier approach, embracing GANs, VAEs, and VAEGANs for level blending. We also integrate an agent-based optimization procedure to ensure the playability of the blended game levels that result from our approach.

The proposed methodology hinges on three essential stages: the integration of game definitions, the implementation of level generation techniques, and optimization through latent vector evolution. In the initial stage, we blend game definitions from two different games. Blending game descriptions demands an organized strategy to blend VGDL game descriptions effectively. The process involves evaluating individual game components in detail, identifying common elements, assessing their compatibility, and merging the game objects, mechanics, rules, and interactions.

Then, in the second stage, we apply level-generation techniques using GANs, VAEs, and VAEGANs. Our methodology's final stage concentrates on optimizing GAN, VAE, and VAE-GAN results through latent vector evolution. We designed a fitness function that evaluates the presence of essential game elements and the playability of the generated levels.

To assess playability, we experimented with two different approaches. First, we implemented the A* algorithm, a commonly used pathfinding method in game navigation, capable of solving the playability of the blended game levels in maze games like Dungeon and Zelda, and ensuring each pellet is accessible in Pacman. By applying this A* heuristic during latent vector evaluation, we can efficiently exclude game levels that are not playable. For solving Sokoban, where the agent must push a box to a hole, we implemented another heuristic function specific to that game. Our second approach is using RL agents to play both games separately. In the RL approach, we assume that if the RL agent is trained to play both games individually, it can then play the blended game. Using RL instead of different A* heuristics is more generalizable because the playability of some game types, such as platformer games, cannot be assessed by A* search heuristics alone.

4.1 Dataset

We conducted experiments with two distinct datasets. The primary dataset is derived from the GVGAI framework, which offers a comprehensive default game level dataset. From this collection, we selected five levels from each game to be blended, resulting in a diverse set of 10 unique levels for blending. Detailed presentation of the selected levels are provided at Appendix A.

Additionally, we manually ported 45 levels into the GVGAI framework, extracting them from the well-known games ‘Sokoban’ and ‘Zelda’. The ‘Sokoban’ levels are sourced from a repository provided by DeepMind [52], while ‘The Legend of Zelda’ levels are obtained from The Video Game Level Corpus (VGLC) repository [53]. Together with the original five levels from each game, we amassed a collection of 100 unique levels to be blended.

The manual porting of levels to the GVGAI framework involves a precise and careful procedure. For instance, the Sokoban levels are originally represented as a grid of characters, where each symbol represents a different game object. A sample Sokoban level is as presented in Figure 1:

```
##########
##########
#######  #
### .# #.#
#     .  #
# #  S S #
# #####  #
##### S.#
### @    #
##########
```

Figure 1: Level Representation of Sokoban in the DeepMind dataset.

In the external dataset’s original Sokoban levels, the characters represent the following game elements: ‘#’ denotes a wall, ‘.’ signifies a goal, ‘S’ symbolizes a box, and ‘@’ designates the player. These are replaced with their corresponding GVGAI representations as presented in Figure 2:

```
wwwwwwwwww
wwwwwwwwww
wwwwwww..w
www.4ww.ww
w.....4..w
w.w..5.5.w
w.wwwww..w
wwwww.554w
www.A....w
wwwwwwwwww
```

Figure 2: Level Representation of Sokoban in GVGAI.

When porting these levels into the GVGAI framework, we converted the characters into their GVGAI counterparts: ‘w’ corresponds to walls, ‘.’ to floors, ‘5’ to boxes, ‘4’ to goals, and ‘A’ to the player.

Finally, we incorporate synthetic data augmentation, commonly known as bootstrapping. This process begins with the original GVGAI dataset. After every 1,000 epochs of model training, we generate 1,000 new levels using the current models. We then filter out unique levels that meet specific playability requirements. The selection criteria for these playable levels are based on seven playability heuristics:

- There is only one player.
- For ‘Zelda’ and ‘Dungeon’, there is only one door.
- Enemies occupy less than 60% of the vacant space.
- In ‘Dungeon’ and ‘Blend’, the avatar can complete the game by either reaching the key and door using an A* algorithm or by collecting all pellets, thereby adhering to the ‘Pacman’ termination rule.
- In ‘Zelda’ and ‘Sokoban’ blends, the player can complete the game by either reaching the key and door using an A* algorithm or by pushing each box into a hole.
- The level has a border of walls to prevent the avatar and enemies from leaving the level.

These unique and playable levels are then added to our training data for the subsequent epochs.

We trained each model with these two different datasets and synthetic data augmentation, contributing to the robustness of our results, and enhancing the evaluation and comparison process of the implemented level generation techniques.

4.2 Level Representation

In the VGDL, a game is characterized by four distinct components. First, a set of sprites represents various entities in the game, each with specific types and properties. An interaction set outlines the game mechanics and the outcomes of different sprite interactions. A termination set defines conditions that result in a win or loss. Fourth, level mapping represents the conversion of ASCII characters into sprites. The current study focuses on blending VGDL games: Pacman with Dungeon and Sokoban with Zelda.

VGDL employs symbolic characters to denote different game elements. These characters are subsequently translated into numeric identity values, facilitating their integration into the General Video Game AI (GVGAI) framework. For instance, Figure 3 demonstrates a level representation of the Dungeon game, as described in VGDL notation.

```
wwwwwwwwwwwwwwwwwwwwww
w.........d...wg....xw
w.....t.......www.wwww
wwwww.wwww...........w
wr...................w
wwwwwwwwwwww.w.w.wwwww
w.ww.w.g...w.w...wwwgw
w..w.wwwww.wmwwwww.w.w
w..f...........f.....w
w.wwww.......wwwwwwwww
w.w..w...f...w.g...g.w
wkw..w.......w...wg..w
wwwwwww......w.......w
1......t.....wwwww.www
w.w............d....gw
w.wwwww..wwww........w
w.w...t..w.g.........2
w...wwgw.wwwwr.......w
w.wwwwww..........lggw
wAwg............u....w
wwwwwwwwwwwwwwwwwwwwww
```

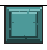
Figure 3: Level Representation for Dungeon in VGDL.

In transforming these level representations into a machine-learning-applicable format, we considered various encoding schemes such as ordinal and binary encoding. Ordinal encoding translates each category into unique integer values, posing a drawback of introducing an artificial order or rank among categories, which does not exist among our tile types [54]. Similarly, binary encoding, while space-efficient, converts categories into binary code and can inadvertently introduce artificial orderings among categories that are inherently unordered. Both binary and ordinal encoding can also complicate the decoding process, especially when the output of the model is not a clear integer or binary value but rather a distribution over possible categories.

One-hot encoding, conversely, represents each category uniquely and treats each category as an independent feature, avoiding any artificial ordinal relationships. This makes one-hot encoding suitable for our data, as tile types in our case are inherently unordered and independent [55]. One-hot encoding also simplifies the decoding process. In situations where the output of the model is a distribution over categories, such as [0.15, 1.2, 0.18, 0.18, 1.01...], we can simply take the max operation to select the most probable category. This is crucial for translating the output of our model back into the original game level representations. Even though one-hot encoding may result in higher dimensionality, with our 16 unique tile types for Dungeon and Pacman and 10 unique tile types for Sokoban and Zelda, this issue is manageable and does not introduce significant computational complexity.

For instance, each unique tile type in the Dungeon-Pacman blend is mapped to a distinct integer value ranging from 0 to 15 (Table 1). These categories are then one-hot encoded into binary vectors of length 16. Each vector contains a '1' in the position corresponding to its respective tile type and a '0' in all other positions, ensuring a robust, mutually exclusive numerical representation of the categorical data. These one-hot encoded vectors serve as inputs for the discriminator network of GAN and the decoder network of VAE and VAEGAN, as these machine learning algorithms perform optimally with numerical input.

Table 1: Tile Representation to One-Hot Encoding Mapping for Dungeon and Pacman games.

Tile Type	Symbol	Index in Encoding Vector	Visualization
Wall	w	0	
Floor	.	1	
Agent	A	2	
Exit	x	3	
Lock	k	4	
Door	m	5	
Trap	t	6	
Fire	f	7	
Gold	g	8	
Power	0	9	
Red Ghost	1	10	
Blue Ghost	2	11	
Pink Ghost	3	12	
Orange Ghost	4	13	
Health	h	14	
Pellet	d	15	

4.3 Blending Game Descriptions

Blended games provide new experiences by merging the distinct mechanics, themes, and structures of their respective games. Using game descriptions, VGDL allows for the blending of game rules and mechanics by offering a formal framework to design and represent various games. This section presents a structured method for combining VGDL game descriptions.

Gow and Corneli [15] proposed a conceptual framework for blending VGDL game descriptions and demonstrated their methodology by creating the 'Frolda' game, a blend of Zelda and Frogger. However, their approach primarily relies on manual intervention, lacking a definitive procedural way to generate new game descriptions. Additionally, they pointed out the difficulty of completely automating this process as blending game descriptions often requires a contextual understanding of game elements and mechanics.

We introduce an approach that involves systematic steps to blend game descriptions while still including some human intervention, particularly in areas where context is important. Our method unfolds in six key stages:

- **Assess compatibility**: Conduct an assessment to ensure that the integration of game components, mechanics, and rules results in a cohesive and functional game.
- **Identify shared elements**: Determine common elements or similarities between the game descriptions, such as shared game objects, interactions, or mechanics that can be smoothly integrated.
- **Merge game objects**: Combine the game objects from the distinct game descriptions, ensuring that the objects and their attributes are compatible. We may modify properties or interactions as needed to maintain consistency.
- **Create a sprite mapping**: Assign new symbols in cases where there is a conflict between symbols.
- **Combine interactions**: Integrate the interactions between game objects from the individual game descriptions, confirming that these interactions are compatible. We eliminate duplicate interactions.
- **Combine termination rules** Integrate game termination rules from both games and eliminate any duplicate rules, if present.

To illustrate our approach, we apply it to the level mappings of Dungeon and Pacman in VGDL, as shown in Figure 4 and Figure 5, respectively. These figures represent the symbolic representation of game elements in VGDL, the foundation of our blending methodology. Level mapping essentially defines the symbolic representation of the various game elements. For instance, in the Dungeon game as depicted in Figure 4, the symbol ‘x’ represents an `exit` on a `floor`, ‘g’ symbolizes `gold` on a `floor`, ‘A’ stands for the `avatar` on a `floor`, and so forth. Similarly, in the Pacman game shown in Figure 5, ‘+’ symbolizes `floor`, ‘0’ denotes `floor power`, ‘.’ signifies `floor pellet`, etc. Therefore, level mappings in this context offer a way to symbolically define each game element and its respective placements.

```
LevelMapping
    x > exit floor
    g > gold floor
    A > avatar floor
    k > key floor
    m > lock floor
    f > firehole floor
    t > groundTrapHole floor
    . > floor
```

Figure 4: Level Mapping of Dungeon.

```
LevelMapping
    + > floor
    0 > floor power
    . > floor pellet
    A > floor hungry
    1 > floor redspawn
    2 > floor orangespawn
    3 > floor bluespawn
    4 > floor pinkspawn
    f > floor fruit
    w > floor wall
```
Figure 5: Level mapping of Pacman.

To conduct an assessment for the compatibility of blending GVGAI games, the process involves several distinct steps. First, each game's individual elements, such as components, mechanics, rules, themes, and objectives, are carefully analyzed to find similarities and differences. This initial analysis helps to identify how the games might align or conflict with each other. Next, an evaluation is made of how these elements could be blended together to create a unified game. This includes understanding how the blending would affect key aspects like gameplay experience, balance between different parts, and engagement of the player. The goal is to see if a cohesive and functional game can be made by combining the identified elements. Finally, it is essential to recognize any constraints or limitations that might block or complicate the blending process. These could include technical issues, conflicts in game mechanics, or differences in the underlying design principles of the games. For example, blending Sokoban, a puzzle game, with Mario, a platformer, might be problematic. The physics and mechanics of the two games are fundamentally different, with Sokoban focusing on precise block-pushing puzzles and Mario emphasizing fluid movement and jumping. Blending these games would likely result in conflicts in gameplay experience and mechanics. On the other hand, blending Pac-Man, a classic maze arcade game, and a dungeon crawler, which involves exploring and navigating through mazes filled with monsters, treasures, and puzzles, could be more compatible. Both games involve navigating through mazes or labyrinthine structures, and their objectives of collecting items and avoiding or defeating enemies can be aligned. Additionally, the grid-based movement and similar game mechanics make the integration more feasible.

Once we contextually decided that games are compatible for blending, in the second step, we identify common elements in both games. Table 2 outlines the common elements in this particular example. To maintain consistency across the blended game, we ensure that identical symbols denote the same game elements. For instance, the 'floor' element in Pacman is transformed to '.' from '+'.

Table 2: Common Sprites between Games.

Pacman	Dungeon	Blend	Visualization
Wall	Wall	Wall	🟦
Floor	Floor	Floor	⬛
Pacman	Avatar	Agent	

We merge sprites in the third step, including the common sprites that are chosen in the 'Identify shared elements' step. In the fourth step, we assign new symbols for conflicting tile representations between the two games; for instance, 'f' represents fire in Dungeon and fruit in Pacman, so in the blend, we assign 'f' for fire and 'h' for fruit. The resulting sprites' mapping is shown in Figure 6. Third and fourth steps are automated with scripts to ensure efficiency.

```
LevelMapping
    x > exit floor
    g > gold floor
    A > agent floor
    k > key floor
    m > lock floor
    f > firehole floor
    t > groundTrapHole floor
    . > floor
    0 > floor power
    d > floor pellet
    A > floor hungry
    1 > floor redspawn
    2 > floor orangespawn
    3 > floor bluespawn
    4 > floor pinkspawn
    h > floor fruit
    w > floor wall
```

Figure 6: Level Mapping of The Dungeon and Pacman Blend.

In the fifth and sixth steps, the process of eliminating possible duplications in the interaction set and the termination set takes place. In VGDL, the interaction set refers to the list of rules that govern the dynamics of the game, such as what happens when a sprite interacts with another. The termination set determines the win/loss conditions for the game based on certain states of sprites. In our example, the interaction set and termination set for the game Pacman (Figure 7) and Dungeon (Figure 9) are distinctly different. However, in the blending process, these rules are combined in a way that eliminates duplications and manages conflicts.

Consider the interaction rule `moving wall > stepBack`, which is present in both games. In the blend, shown in Figure 8, this rule is retained only once, demonstrating the elimination of duplication.

In another case, in the original Pacman game, `power hungry > killSprite` represents an interaction where a power pellet (symbolized as `power`) causes a `hungry` ghost to be `killed` or removed from the game. In contrast, the Dungeon game has no such interaction. In the blended game, this interaction rule is preserved without modification, as there is no conflicting rule in the Dungeon game that needs to be addressed.

The termination sets are also merged in a similar way. For example, the termination rule `SpriteCounter stype=food win=True` of the Pacman game and the rule `SpriteCounter stype=exit win=True` of the Dungeon game are both retained in the blended game as shown in Figure 8. The incorporation of these termination rules means that the game will finish either when all the `food` sprites are consumed, which is similar to the Pacman game, or when the `exit` sprite is reached, like in the Dungeon game.

Thus, by eliminating duplications and resolving conflicts, in the fifth and sixth steps, we ensure that the blended game possesses a coherent set of rules derived from both original games.

```
TerminationSet
    SpriteCounter stype=food   win=True
    SpriteCounter stype=pacman win=False
InteractionSet
    moving wall > stepBack
    pacman EOS  > wrapAround
    ghost EOS ghost > stepBack
    power hungry  > killSprite
    hungry ghost  > killSprite scoreChange=-1
    power pacman > killSprite scoreChange=10
    pellet pacman > killSprite scoreChange=1
    fruit pacman > killSprite scoreChange=5

    hungry power > transformToAll
    hungry power > transformToAll
    hungry power > transformToAll
    hungry power > transformToAll

    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    killSecond=True

    hungry power > addTimer
    hungry power > transformTo stype=powered
    ghost powered > killSprite scoreChange=40
```
Figure 7: Pacman Termination Set and Interaction Set.

```
TerminationSet
    SpriteCounter stype=food win=True
    SpriteCounter stype=agent win=False
    SpriteCounter stype=exit win=True
InteractionSet
    gold avatar > killSprite scoreChange=5
    key avatar
        > collectResource scoreChange=50
    avatar wall > stepBack
    lock avatar
        > killIfOtherHasMore resource=key
    avatar lock > stepBack
    exit avatar
        > killSprite scoreChange=200
    avatar fireEnd
        > subtractHealthPoints
    fireEnd avatar wall resources doors
        > killSprite
    avatar groundTrap
        > subtractHealthPoints value=2

    moving wall > stepBack
    avatar EOS  > stepBack
    ghost EOS ghost > stepBack

    power hungry  > killSprite
    hungry ghost  > killSprite scoreChange=-1

    power avatar > killSprite scoreChange=10
    pellet avatar > killSprite scoreChange=5
    fruit avatar > killSprite scoreChange=10

    hungry power > transformToAll
    hungry power > transformToAll
    hungry power > transformToAll
    hungry power > transformToAll

    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    killSecond=True
    hungry power > addTimer
    stypeTo=orangeOk killSecond=True

    hungry power > addTimer
    hungry power > transformTo stype=powered

    ghost powered > killSprite scoreChange=40
```
Figure 8: Termination Set and Interaction Set After Combining Dungeon and Pacman.

```
TerminationSet
    SpriteCounter stype=avatar win=False
    SpriteCounter stype=exit win=True
InteractionSet
    gold avatar > killSprite scoreChange=1
    key avatar  > collectResource
    avatar wall boulderHole > stepBack
    lock avatar
            > killIfOtherHasMore resource=key
    avatar lock > stepBack
    exit avatar
            > killSprite scoreChange=10
    avatar groundTrap
            > subtractHealthPoints value=2
```

Figure 9: Dungeon Termination Set and Interaction Set.

4.4 Blending Game Levels With Generative Networks

Generative networks, like GANs and VAEs, can create realistic game levels when trained on game data. Specifically, VAEs can mix elements from two different games to create a new level. This mixing is shown in Figure 10. In our study, we used three networks: GANs, VAEs, and VAEGANs, to mix levels from different games. Before feeding game levels into these networks, we make some changes, as explained in section 4.3, to ensure levels have common elements like actors and walls. Then, we adjust the size of each level using wall tiles. For example, after this adjustment, Dungeon and Pacman game levels are 32x32 in size, while Zelda and Sokoban game levels are 16x16.

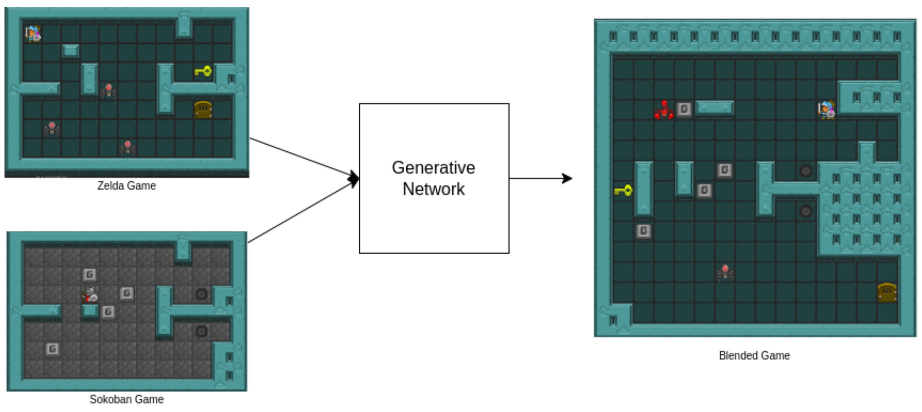


Figure 10: Process of Level Blending.

Conditionality is introduced by appending the number of tile types in the level as a conditional variable. For instance, consider the following game level in Figure 11:

```
wwwwwww
w..0..w
w.w1w.w
w...A.w
wwwwwww
```

Figure 11: Example Game Level.

In this level, we can identify different tile types such as walls (w), empty spaces (.), objects labeled ‘0’ and ‘1’, and the avatar (A). By analyzing the distribution, we append the count of each tile type as a conditional variable. In this example, the conditionality may include 14 walls, 16 empty spaces, 1 avatar, 1 object of type ‘0’, and 1 object of type ‘1’.

This additional information refines the generator’s output, enabling it to understand the context of the game level. Consequently, the model generates levels representing the original game levels’ distribution. This approach has been employed in the context of conditional GANs, following the approach by Torrado et al. [56], and in the conditional VAE and VAE-GAN as demonstrated by Sarkar et al. [18] and Zakaria et al. [57]. Unlike post-optimization methods, where the generated levels are further refined or adjusted after the initial generation, introducing conditionality allows the generator to create levels that more accurately reflect the original design characteristics from the beginning of the generation process.

To improve our training, we used synthetic data augmentation, also known as bootstrapping. Both Torrado et al., with 5 original levels, and Zakaria et al., with 12 original levels, used bootstrapping to train and generate new game levels [56, 57]. They tried both with and without bootstrapping and found that bootstrapping method, combined with networks like GANs and VAEs, produces varied levels. Following this methodology, our procedure is initiated with the foundational GVGAI dataset. After every set of 1,000 training epochs, the current model produces an additional 1,000 new levels. From these, only the distinct levels that meet specific playability criteria are added to the training data. The selection of these playable levels is based on seven key heuristics:

- A singular player per level.
- Only one door in the ‘Zelda’ and ‘Dungeon’ games.
- Enemies should not occupy more than 60% of the available space.
- For ‘Dungeon’ and ‘Blend’, the avatar’s win condition can be either reaching the key and door via the A* algorithm or collecting all pellets, adhering to the ‘Pacman’ termination rule.
- In blends of ‘Zelda’ and ‘Sokoban’, victory can be achieved by either reaching the key and door using the A* algorithm or ensuring each box finds its way into a hole.
- The level is encapsulated by a wall border to inhibit avatars and enemies from departing the level.

Levels that fulfill these criteria are integrated into our training data for subsequent epochs.

4.4.1 Blending Game Levels with GAN

This study employs Generative Adversarial Networks (GANs) to generate blended game levels derived from pre-existing level representations. Specifically, we leverage Deep Convolutional Generative Adversarial Networks (DCGANs), a variant of GAN models, trained using the Wasserstein GAN (WGAN) algorithm proposed by Arjovsky et al. [21]. The WGAN algorithm has demonstrated its effectiveness in the procedural generation of Super Mario Bros. levels.

Our DCGAN architecture is based on modifications suggested by Volz et al. [38], which include strided convolutions in the discriminator and fractional-strided convolutions in the generator, in line with the WGAN structure. Batch normalization is employed in both the generator and discriminator following each layer to stabilize learning and maintain the quality of generated samples. Unlike the original WGAN, our design uses Rectified Linear Unit (ReLU) activation functions in all generator layers except the output layer, which employs a Hyperbolic Tangent (Tanh) function.

Self-attention mechanisms are integrated into both the generator and discriminator to capture and utilize global dependencies in data. This approach is inspired by Conditional Embedding Self-Attention Generative Adversarial Networks (CESAGANs), which combines conditionality and self-attention for robust results [58].

Our model employs an adaptive training schedule. The discriminator is updated more often at the beginning of training and at regular intervals, fostering healthy competition between the networks and enhancing overall performance. Moreover, parameter clamping is applied to the discriminator network for learning stability and to prevent mode collapse.

We use the Adam optimizer for training both the generator and discriminator networks due to its efficiency and suitability for problems with large data or parameters [59]. Noise is introduced as part of the input to both networks, aiding in the generation of diverse samples and mitigating overfitting.

In the GAN training process, each integer tile is converted into a one-hot vector, creating 16 channels of size 32 x 32 for the Dungeon-Pacman blends and 10 channels of size 16 x 16 for the Zelda-Sokoban blend. Following training, the generator's output is converted into an integer representation using the argmax operator. The result is decoded into a VGDL game representation.

4.4.2 Blending Game Levels with VAE

Alongside GANs, we employ VAEs to generate game levels using pre-existing level representations in this research. VAEs offer us the benefit of modeling our data within a latent space where proximate points represent similar game levels.

The architecture of our VAE model consists of two primary components: an encoder and a decoder. The encoder's function is to transform an input game level into a corresponding latent vector, while the decoder accepts a latent vector as input and produces a new level.

Essentially, the encoder distills the crucial characteristics of the input data, and the decoder uses these features to construct new data resembling the original.

Our VAE model follows the same principles as our GAN model. The encoder integrates strided convolutions, LeakyReLU activations, and additional layers as necessary. The output of the encoder comprises two vectors of length 'nz' (indicating the latent space's dimensionality), representing the mean and the log-variance of the Gaussian distribution. The latent vector is sampled from this distribution.

Conversely, the decoder incorporates a structure similar to the encoder but in reverse order. It employs transpose convolutions to increase the spatial resolution of the input alongside ReLU activations. In addition, the output layer utilizes a ReLU activation function to align the range of the output data with that of the input data.

Given an input, the model passes it through the encoder to obtain a mean and a log-variance vector. A latent vector is then sampled from the Gaussian distribution defined by these vectors. Finally, the decoder reshapes and propagates this latent vector to generate the output data. This reshaping operation ensures that the latent vector is appropriately formatted for the transpose convolutions in the decoder.

The VAE model's training procedure involves backpropagating the reconstruction loss, assessing the decoder's capacity to replicate the input data and the Kullback-Leibler (KL) divergence between the learned Gaussian distribution and the standard normal distribution. This additional measure encourages a well-structured latent space within the model and improves the quality of the generated data.

Upon the completion of training, the model can accept random vectors sampled from the standard normal distribution to generate new game levels. These levels are converted into a VGDL game representation for further use.

4.4.3 Blending Game Levels with VAEGAN

This study used a VAEGAN to create blended game levels by combining level designs from existing Mario, Dungeon, Zelda, and Sokoban games. The architecture of our VAEGAN integrates features of both the GAN, as described by Arjovsky et al. [21], and the VAE models discussed above. Furthermore, VAEGAN is using the advantages of both VAE and GAN models. For example, it combines VAE's ability to reconstruct losses with GAN's adversarial training component.

The VAEGAN model comprises three primary components: an encoder, a generator/decoder, and a discriminator. The encoder maps the input game level into a corresponding latent vector, similar to our VAE model. Conversely, the generator/decoder mirrors the generator in our GAN model, accepting a latent vector input and producing a new game level. Finally, the discriminator is tasked with distinguishing genuine game levels from the ones generated.

Our VAEGAN aligns with the principles of the DCGAN, incorporating strided convolutions in the discriminator and fractional-strided convolutions in the generator/decoder. We also apply batch normalization in both the generator/decoder and discriminator following each layer to stabilize learning and maintain the quality of the generated samples. The generator/decoder employs Rectified Linear Unit (ReLU) activation functions across all its layers, including the output layer, which traditionally uses the Hyperbolic Tangent (Tanh) activation function. In contrast, the discriminator and encoder use LeakyReLU activation in all their layers.

The training process of the VAEGANs is similar to that of the GANs and VAEs. First, each integer tile is expanded into a one-hot vector, resulting in 16 channels of size 32 x 32 for the Dungeon-Pacman blend for the discriminator. Similarly, we use 10 channels of size 16 x 16 for the Zelda-Sokoban blend. The generator/decoder, in contrast, accepts a latent vector of length 128 as input.

The training process consists of two distinct stages. The first stage adopts the VAE approach, minimizing the reconstruction loss and the KL divergence. The second stage follows the GAN approach, training the generator/decoder and the discriminator adversarially. This dual-stage training procedure allows the model to exploit the benefits of both VAE and GAN.

During the evolutionary process, the final output of the generator/decoder, a three-dimensional tensor of dimensions 16 x 32 x 32 for the Dungeon and Pacman blend and 10 x 16 x 16 for the Zelda and Sokoban blend, is converted into a 2D representation with taking the index of the maximum value in the first dimension. This 2D representation can be decoded into a VGDL game representation.

4.5 Optimization of Latent Vectors

Generating game levels with our GAN, VAE, and VAEGAN models requires the optimization of latent vectors. A latent vector, within the context of these generative models, is a condensed, high-dimensional vector that encapsulates essential characteristics of the dataset. It functions as the input for the generator or decoder module, leading to an output that closely reflects the learned data distribution.

Optimizing the latent vector that is provided to the generator of GANs or decoder of VAEs lets us generate levels with a distribution of tiles that we want to generate. We exploit this concept to ensure playability and the necessary inclusion of game elements in generated levels. We devised a two-step fitness function for optimizing the latent vector, as illustrated in Figure 12. The first step of this function focuses on ensuring that the essential game elements are included in the generated level. For example, it checks for the presence of an agent sprite, an exit sprite for Dungeon and Zelda games, a minimum of one pellet and one ghost for Pacman games, and a minimum of one box and one hole for Sokoban games.

The subsequent step of the fitness function uses an A* path-finding algorithm to evaluate the navigability of the generated level, ensuring a viable path to the level's end. Two solvers incorporate A* path-finding algorithms with different heuristics to recognize if there is an optimal path from start to finish. Specifically, for Dungeon, Zelda, and Pacman games, we utilize the MazeSolver, which confirms that the agent can reach all pellets. In addition, a distinct solver, SokobanSolver, is employed for Sokoban games. This solver factors in both the position of the player and the box to maneuver the box to the exit point.

Here, we define methods for heuristic cost estimation, distance measurement, and the computation of adjacent, reachable nodes (neighbors) for a given node. We utilize the A* algorithm with different heuristics to create separate solvers for various games such as Dungeon, Pacman, Sokoban, and Zelda. For Dungeon, the A* heuristic is employed to search for a viable path between actor-key and actor-door. In Pacman, the heuristic ensures that the actor can reach all the pellet objects by eating all pellets required to finish the game. In Sokoban, the heuristic helps in identifying a feasible path from each box to any hole, as well as from the actor to the edge of the box. In Zelda, the algorithm is utilized to inspect if the actor has a possible route to the exit door. Pseudocode for the MazeSolver is provided in Figure 14, and for the SokobanSolver is given in Figure 13. A* heuristic for MazeSolver looks for whether there is a viable path between two given objects in the game. We look for a viable path between actor-key and actor-door for the Dungeon game. In Pacman, we check whether the actor can reach all the pellet objects by eating all pellets required to finish the game. For Zelda, we inspect if the actor has a possible route to the exit door. On the other hand, in SokobanSolver, the 'neighbors' method is adapted to accommodate the positions of both the player and the box. Therefore, to analyze the solvability of Sokoban, we should make sure that there is a viable path from each box to any hole. Additionally, we need a viable path at each step, from actor to edge of the box, in the opposite direction of a route from the selected box to the corresponding hole. It's important to note that while existing Sokoban solvers can be utilized, we have chosen to create an A* heuristic function to maintain consistency and the same concept throughout the study.

Through the utilization of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm, we determine the best levels that meet the requirements defined by our fitness function. In this way, our optimization strategy guarantees that the generated levels not only contain necessary game elements but are also playable.

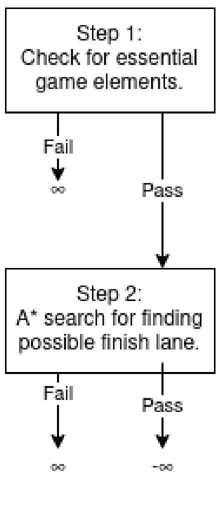

Step 1:
Check for essential game elements.
Fail
∞
Pass
Step 2:
A* search for finding possible finish lane.
Fail
Pass
∞
-∞

Figure 12: Process of Fitness Function Evaluation with A* solver.

```
class SokobanSolver:

    def heuristic_cost_estimate(n1, n2):
        (bx1, by1), _ = n1
        (bx2, by2), _ = n2
        dx = bx2 - bx1
        dy = by2 - by1
        return sqrt(dx**2 + dy**2)

    def distance_between( n1, n2):
        _, (px1, py1) = n1
        _, (px2, py2) = n2
        dx = px2 - px1
        dy = py2 - py1
        return sqrt(dx**2 + dy**2)

    def neighbors( node):
        box, player = node
        b_x, b_y = box
        p_x, p_y = player
        directions = [
            (0, 1),  # up
            (0, -1),  # down
            (-1, 0),  # left
            (1, 0)  # right
        ]
        player_neighbors = []
        box_neighbors = []
        for dx, dy in directions:
            player_node = (p_x+dx, p_y+dy)
            box_node = (b_x+dx, b_y+dy)
            if is_valid(player_node):
                player_neighbors
                    .append(player_node)
            if is_valid(box_node):
                box_neighbors.append(box_node)
        return player_neighbors, box_neighbors

    def is_valid(self, node):
        return point is within the maze bounds
            and not a wall
```

Figure 13: Pseudocode for SokobanSolver.

```
class MazeSolver:

    def heuristic_cost_estimate(n1, n2):
        (x1, y1) = n1
        (x2, y2) = n2
        dx = x2 - x1
        dy = y2 - y1
        return sqrt(dx**2 + dy**2)

    def neighbors(node):
        x, y = node
        directions = [
            (0, 1),  # up
            (0, -1),  # down
            (-1, 0),  # left
            (1, 0)  # right
        ]
        neighbors = []
        for dx, dy in directions:
            new_node = (x+dx, y+dy)
            if is_valid(new_node):
                neighbors.append(new_node)
        return neighbors

    def is_valid(node):
        return point is within the maze bounds
            and not a wall
```

Figure 14: Pseudocode for MazeSolver.

4.5.1 Using Reinforcement Learning Agents For Optimization

Since both Dungeon, Pacman blend and Sokoban, Zelda blend are games with maze-like characteristics A* agents are applicable for solving whether the game is playable or not. However, in different game types such as platformers, we need more generic solutions to test the playability of resulting games. Therefore we researched Reinforcement Learning Agents.

In this thesis, we use the Advantage Actor-Critic (A2C) reinforcement learning algorithm, specifically the implementation from the Open AI Baselines, along with the GVG-AI Gym framework. The A2C algorithm is known for its ability to achieve a balance between computational efficiency and policy efficacy, making it ideal for multi-game environments. We used the configuration of Justesen et al. to use multiple parallel workers, a step size of tmax = 5, no frame skipping, and a constant learning rate of 0.007 with the RMS optimizer [51].

The objective of A2C training is to create an agent capable of interacting with these two distinct game environments and learning how to make decisions that maximize its cumulative reward within each game. Here, the agent's ability to finish each game–Pacman and Dungeon –determines the cumulative reward. Therefore, greater rewards indicate that the agent is able to finish levels.

The A2C algorithm operates by simultaneously training two distinct networks: an actor network that makes decisions and a critic network that evaluates the actor's decisions. The actor's responsibility is to determine the actions taken by the agent in each environment, PacMan and Dungeon. In contrast, the critic provides the actor with feedback in the form of value estimates, which represent the expected future rewards for each action. This dual feedback guides the agent towards actions that result in greater rewards in both games.

The neural network has the same architecture as in Mnih et al., with three convolutional layers and a single fully-connected layer [47]. The game state, represented as a 2D array of sprite representations, is processed by these convolutional layers, which extract essential features. The fully integrated layer then uses these characteristics to make decisions for both Pacman and Dungeon. This hierarchical representation allows the agent to make complex decisions that enhance its performance in both games.

The agent is trained at a constant rate over multiple timesteps. The selection of the learning rate and other hyperparameters was based on previous research and preliminary experiments. RMSProp, a popular optimizer for managing sparse gradients and dynamically adjusting learning rates, is used to regulate training.

During training, the model interacts with both game environments, collects experience, and uses it to update its policy and value networks continuously. This procedure is repeated until the agent reaches an acceptable performance level or training resources are exhausted. Then we used a trained RL agent within the new step in the previously defined fitness function for CMA-ES. The new two-step fitness function for optimizing the latent vector, as illustrated in Figure 15. The first function is the same as before but we replace the A* search step with a new step that uses an RL agent. Eventually, a new two-step fitness function is defined. The first stage validates necessary game elements within the level. This includes verifying crucial sprites for Dungeon, Zelda, Pacman, and Sokoban games. The second stage runs a Reinforcement Learning (RL) agent. If the agent successfully completes the game, the level is selected. The RL agent's performance acts as a measure of the level's playability.

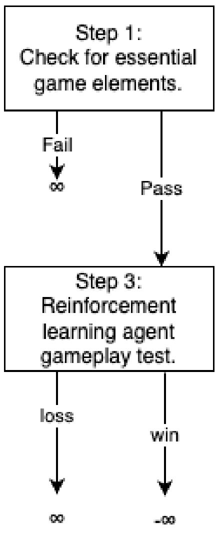

Step 1:
Check for essential game elements.
Fail
∞
Pass
Step 3:
Reinforcement learning agent gameplay test.
loss
win
∞
-∞

Figure 15: Process of Fitness Function Evaluation with RL agents.

CHAPTER 5

EXPERIMENTS AND RESULTS

We have performed experiments to measure the effectiveness of our proposed methodology by employing the GVGAI framework as the testing bed. We chose four games from the GVGAI sample games collection for this purpose. The games are Dungeon, Pacman, Zelda, and Sokoban. Each game has its own set of gameplay mechanics and objectives to win. In the Dungeon and Pacman games, the actor must navigate through mazes and complete specific objectives such as eating all the edible items in Pacman and reaching an exit in Dungeon. In the Zelda and Sokoban games, actors must solve puzzles by finding keys, unlocking doors, and moving objects.

As a part of our experimental procedure, we have attempted to blend two pairs of games using our proposed methodology. The first blend combines Dungeon and Pacman, while the second combines Sokoban and Zelda. We use GAN, VAE, and VAEGAN models to generate these blended game levels. To ensure consistent inputs to our models, we used five sample game levels as references and resized or added padding to the games as needed to maintain a standardized grid size. In addition to the Baseline GVGAI dataset, we used an Extended dataset with an additional 45 levels, which further enriches our training data. Furthermore, we also implemented bootstrapping within the model in separate experiments to explore different techniques. We trained these models over 10,000 epochs while taking regular snapshots at every 1000 epochs and we selected the best models as described in Section 5.1.

We created game levels by using the CMA-ES optimization algorithm. First, we optimized the latent vector with CMA-ES to generate the levels. Then we eliminated any repeated levels to ensure each level was unique. We generated 1,000 sample games from all models.

Lastly, we measured the effectiveness of our approach using three main evaluation metrics: Novelty, Tile Density, and Diversity. Novelty measures the distinctness of the levels compared to the training data. Tile Density inspects the complexity of the generated game levels. Finally, diversity assesses the variation among the generated samples.

In the subsequent subsections, we discuss each aspect in detail: Model Selection, Novelty of Generated Levels, Tile Density of Generated Levels, Diversity Analysis of Generated Levels, Comparison of Models, and Analysis of Levels Generated By RL Based Latent Vector Optimization. Each subsection thoroughly explains the evaluation process, covering the experimental setup, methodology, and results.

5.1 Model Selection

The model selection process for the Generative Adversarial Network (GAN), Variational Autoencoder (VAE), and VAE-GAN was executed by considering three specific scenarios: the baseline GVGAI dataset, an extended dataset (only for the Sokoban Zelda blend), and a model with bootstrapping. Each model was trained for 10,000 epochs, with snapshots captured every 1,000 epochs, using various hyperparameter configurations.

Upon the completion of training, 1,000 samples were generated from each snapshot without latent vector optimization. The evaluation of these samples was based on two metrics: the uniqueness ratio and the playability ratio. The uniqueness ratio quantifies the level of differentiation among the generated levels, with a value of 1 indicating complete uniqueness. The playability ratio is a quantifiable measure that evaluates the levels in terms of their playability.

The selection process aimed to identify the model snapshot that produced levels with the highest playability ratio provided that the uniqueness ratio remained greater than 0.9.

The findings of our model evaluation for the GAN models trained on the baseline dataset are exemplified in Table 3 below. Here, ‘ngf’ denotes the number of generator feature maps, ‘nz’ refers to the size of the latent vector, and ‘epoch’ indicates the specific stage of training evaluated.

Table 3: Model evaluation results for GAN trained on Baseline GVGAI Dataset for Dungeon Pacman blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	4000	1	0.218
32	64	9000	1	0.208
32	64	8000	1	0.204
64	64	3000	1	0.198
64	64	2000	1	0.194
64	256	5000	1	0.188
64	64	1000	1	0.183
64	128	9000	1	0.178
64	128	10000	1	0.173
64	64	7000	1	0.168
32	64	1000	1	0.163
64	128	6000	1	0.158
64	256	2000	0.346	0.153

The table with label 3 shows the results of our evaluation, organized by uniqueness and playability ratios. We’ve removed entries with low uniqueness scores to only include those that meet our standards. The table is sorted so that the higher uniqueness and playability scores come first. From this sorted table, we choose the top entry, as it has the best combination of uniqueness and playability. This way, the table helps us easily identify the best configuration for our model. Tables detailing the evaluation results for other models, including VAE and

VAE-GAN trained on different blends, can be found in the Tables B1- B14 in the Appendix B.

5.2 Novelty of Generated Levels

Measuring how unique the generated levels are compared to the levels used for training is very important. Figuring out how different levels are from each other can be hard. To solve this problem, we use the Hamming Distance, a method that counts how many parts are different between two things. This method has been used in a similar way by other researchers like Torrado et al. and Zakaria et al. [56, 57].

The Hamming Distance was selected not just because of its compatibility with the string-based level representations of GVGAI games, but also for its common usage in the literature. It offers a straightforward way to understand and measure differences: by counting how many tiles are different between two levels. Furthermore, Hamming Distance allows for quick calculations, a critical advantage when dealing with numerous levels. Even though it has certain limitations, like possibly not capturing deeper level similarities and two dimensional structures, its benefits made it a fitting choice.

To elucidate the concept of Hamming distance, we can observe two example Sokoban levels. In Figure 16, we have Sokoban Level 1, and in Figure 17, we have Sokoban Level 2.

```
wwwwwww
w..0..w
w.w1w.w
w...A.w
wwwwwww
```

Figure 16: Sokoban Level 1.

```
wwwwwww
w..0A.w
w.w1w.w
w.....w
wwwwwww
```

Figure 17: Sokoban Level 2.

In these example levels, '.' indicates an empty space, 'w' stands for a wall, '0' and '1' are boxes, and 'A' represents the agent. The Hamming distance between these levels is 2, owing to the two positions where they are different. Such a difference measurement assists in quantifying how much a generated level deviates from training levels.

By calculating the Hamming distance from levels in the training dataset for each procedurally generated level, we can determine the minimum Hamming distance, which serves as our measure of novelty.

Typically, the Hamming Distance is an integer value indicating the number of character differences in a string. In our approach, we normalize this value by the size of the string, ensuring the novelty score lies within [0,1]. A lower score indicates higher similarity. Accordingly, we defined the Novelty Score equation as:

$$N(\mathbf{l}gen, \mathbf{L}train) = \frac{\min_{\mathbf{l}train \in \mathbf{L}train} d_H(\mathbf{l}gen, \mathbf{l}train)}{|\mathbf{l}_{gen}|} \tag{10}$$

Where $\mathbf{l}gen$ represents a generated level, $\mathbf{L}train$ denotes the set of training levels, and $d_H(\mathbf{l}gen, \mathbf{l}train)$ symbolizes the Hamming distance—counting positions with different tiles in the generated and a particular training level. By dividing this count by the total positions in the generated level, denoted by $|\mathbf{l}_{gen}|$, the score gets normalized to [0,1].

The Novelty Score illustrates the dissimilarity of a generated level from the closest training level. A smaller Novelty Score suggests a greater similarity to a training level, whereas a higher score indicates a distinct, novel level.

Table 4 presents the novelty scores of generated levels for the Sokoban and Zelda blend using different models and datasets.

In the Baseline dataset, the VAE and VAEGAN models produce levels with similar minimum, 10%, 90%, and maximum novelty scores. However, the GAN model seems to generate more levels that closely resemble those in the training set, evident from its lower average novelty score of 0.15. The average novelty score for VAE is 0.18, slightly lower than VAEGAN's 0.19. Concerning the variability in scores, GAN has a slightly wider spread, with a standard deviation of 0.07, compared to 0.06 for both VAE and VAEGAN.

For the levels generated using the extended dataset, the novelty scores for all three models are higher than those from the Baseline dataset. VAEGAN has the highest average novelty score of 0.28, followed by VAE with 0.25, and GAN at 0.23. As for the distribution of scores, GAN exhibits a wider spread, with a standard deviation of 0.09, compared to the 0.07 and 0.06 exhibited by VAE and VAEGAN respectively.

With the Bootstrapping dataset, the produced novelty scores align closely with the Baseline for GAN, with an average of 0.18. The average scores for VAE and VAEGAN are 0.21 and 0.23 respectively, with both outperforming GAN. Similar to other datasets, GAN shows a wider distribution in scores, with a standard deviation of 0.07, while VAE and VAEGAN maintain a standard deviation of 0.05.

Table 5 illustrates the novelty scores of generated levels for the Dungeon and Pacman blend using different models and datasets.

For the Baseline dataset, VAE and VAEGAN models create almost identical minimum, 10%, 90%, and maximum novelty scores. However, GAN-generated levels tend to follow the training set more, as seen from its lower average novelty score of 0.12. The average scores for VAE and VAEGAN are 0.20 and 0.21, respectively. In terms of variability, GAN holds a slightly larger spread with a standard deviation of 0.07, while VAE and VAEGAN exhibit a consistent standard deviation of 0.06.

Table 4: Novelty scores of generated levels for the Sokoban and Zelda blend.

Training Data	Model	Min	10%	90%	Max	Avg	Stdev
Baseline Dataset	GAN	0.01	0.03	0.15	0.36	0.15	0.07
	VAE	0.02	0.08	0.23	0.30	0.18	0.06
	VAEGAN	0.02	0.07	0.22	0.31	0.19	0.06
Extended Dataset	GAN	0.02	0.14	0.34	0.47	0.23	0.09
	VAE	0.03	0.20	0.33	0.37	0.25	0.07
	VAEGAN	0.03	0.19	0.32	0.38	0.28	0.06
Bootstrapping	GAN	0.01	0.05	0.18	0.41	0.18	0.07
	VAE	0.02	0.09	0.24	0.33	0.21	0.05
	VAEGAN	0.02	0.08	0.25	0.36	0.23	0.05

Regarding the Bootstrapping dataset, the levels' novelty scores are higher than those from the Baseline dataset. VAE and VAEGAN have average scores of 0.24 and 0.26 respectively, both outpacing GAN's average of 0.18. GAN continues to display a more extensive distribution, with a standard deviation of 0.08, while VAE and VAEGAN sustain a standard deviation of 0.06.

Table 5: Novelty scores of generated levels for the Dungeon and Pacman blend.

Training Data	Model	Min	10%	90%	Max	Avg	Stdev
Baseline Dataset	GAN	0.03	0.05	0.17	0.24	0.12	0.07
	VAE	0.04	0.10	0.26	0.35	0.20	0.06
	VAEGAN	0.04	0.11	0.27	0.36	0.21	0.06
Bootstrapping	GAN	0.04	0.07	0.20	0.29	0.18	0.08
	VAE	0.05	0.12	0.28	0.29	0.24	0.06
	VAEGAN	0.05	0.13	0.29	0.43	0.26	0.06

5.3 Tile Density of Generated Levels

Tile density, indicating the count of non-background tiles within a game level, is a commonly adopted metric in game level generation literature due to its capacity to convey a level's structure and complexity. Higher tile density levels generally suggest more gameplay elements or obstacles, whereas lower densities imply more open space.

In their research, Sarkar et al. [46] defined 'Density' as the count of non-background or non-path tiles within a segment, providing a clear metric to capture the essence of gameplay elements present. Similarly, in another study [44], Sarkar used 'Density' to represent the solid tiles within a segment, showcasing its versatility.

Torrado et al. [56] took a different approach by examining tile distributions, offering insights into the variability and diversity of generated game levels. Their observations highlighted the differences in the abilities of various generative models to produce diverse levels.

Following the previous use of tile densities in literature, in our study we intend to analyze tile densities from a dual perspective: firstly, to understand the complexity of the generated levels, and secondly, to assess the capability of the generative networks. This approach provides a balanced view that not only presents a level's spatial complexity but also shows the effectiveness of the employed networks.

Upon examining Table 6, we make following observations for the tile densities of generated levels for the Sokoban and Zelda blend using different models.

For the baseline data, the VAEGAN consistently shows the highest average wall tile count at 170, surpassing both the VAE and GAN models which have averages of 162 and 145 respectively. Conversely, when analyzing background tiles, the VAEGAN generates the fewest with an average of 65, while the VAE and GAN models produce averages of 82 and 100, respectively. This demonstrates that VAEGAN tends to produce denser levels with fewer open spaces in the baseline model compared to the other techniques.

For the levels generated from the extended dataset, all models display similar trends in terms of wall tiles, with average counts ranging from 147 (GAN) to 172 (VAEGAN). In the case of background tiles, the GAN model has the highest average at 102, while the VAE and VAEGAN models have slightly lesser averages of 84 and 67 respectively.

Lastly, in the bootstrapping data, VAEGAN continues to display a higher tendency towards denser levels with an average wall tile count of 171, whereas the VAE and GAN models remain in the middle and lower end with averages of 163 and 146, respectively. Background tiles follow a similar trend observed in the previous datasets, with GAN having the highest average of 101, and VAEGAN producing the least with an average of 66.

Table 6: Tile densities of generated levels for Sokoban and Zelda blend.

Data	Model	Tile	Min	10%	25%	75%	90%	Max	Avg	Stdev
Baseline Dataset	GAN	Wall	80	90	130	155	180	190	145.2	35.4
		Background	80	95	107	115	120	125	100.5	15.3
	VAE	Wall	80	100	140	165	185	190	162.3	40.1
		Background	70	75	80	90	100	110	82.4	12.6
	VAEGAN	Wall	80	105	155	175	190	190	170.5	45.2
		Background	60	63	70	80	90	100	65.7	12.3
Extended Dataset	GAN	Wall	82	92	135	157	182	188	147.6	36.5
		Background	78	93	105	113	118	123	102.3	15.2
	VAE	Wall	82	102	142	162	184	188	164.1	41.4
		Background	68	73	78	88	98	108	84.6	13.5
	VAEGAN	Wall	82	107	152	172	188	188	172.4	42.3
		Background	58	61	68	78	88	98	67.1	13.2
Boot-strapping	GAN	Wall	80	91	131	156	181	189	146.7	37.3
		Background	79	94	106	114	119	124	101.2	16.1
	VAE	Wall	80	101	141	166	186	190	163.4	41.2
		Background	69	74	79	89	99	109	83.7	13.4
	VAEGAN	Wall	80	106	154	174	189	190	171.3	43.1
		Background	59	62	69	79	89	99	66.5	14.2

Upon examining Table 7, we list following observations about the tile densities of generated levels for the Dungeon and Pacman blend using different models, taking into account that this game blend is approximately 2.5 times larger than the Sokoban and Zelda blend.

In the baseline data, the GAN model exhibits the tightest distribution among percentiles for wall tiles with a standard deviation of 30.3, suggesting consistency in its level generation. On the other hand, the VAEGAN and VAE models display larger standard deviations of 50.6 and 45.5, respectively, implying more varied level designs. When considering average tile counts, the VAEGAN model leads in wall tiles with an average of 280.5, with VAE at 275.3 and GAN at 260.2, indicating significant differences in the context of the larger Dungeon and Pacman blend.

In the bootstrapping data, the differences between the models begin to converge, with the GAN model showing a broader distribution than in the baseline with a standard deviation of 33.4 for wall tiles but still tighter than its counterparts (VAEGAN at 44.5 and VAE at 42.8). The VAEGAN and VAE models continue to produce levels with a wide range of tile densities, reflecting the diverse influences from the bootstrapping process, with average wall tile counts at 279.4, 272.7, and 262.3 for VAEGAN, VAE, and GAN, respectively.

Table 7: Tile densities of generated levels for Dungeon and Pacman blend.

Data	**Model**	**Tile**	**Min**	**10%**	**25%**	**75%**	**90%**	**Max**	**Avg**	**Stdev**
Baseline Dataset	GAN	Wall	190	200	220	290	310	315	260.2	30.3
		Empty	220	242	262	300	312	315	271.1	30.4
	VAE	Wall	185	205	230	305	316	320	275.3	45.5
		Empty	220	246	263	303	318	320	278.2	33.4
	VAEGAN	Wall	180	210	240	310	315	320	280.5	50.6
		Empty	220	236	250	306	320	320	275.4	35.2
Bootstrapping	GAN	Wall	192	202	222	293	313	317	262.3	33.4
		Empty	222	244	257	302	316	320	273.5	31.6
	VAE	Wall	188	209	232	302	319	320	272.7	42.8
		Empty	221	247	259	304	316	320	276.1	32.3
	VAEGAN	Wall	183	213	243	309	315	320	279.4	44.5
		Empty	220	237	251	307	313	318	277.6	34.7

5.4 Diversity Analysis of Generated Levels

The analysis of diversity among the generated levels is crucial for our experiments. We have chosen the same metric used in Section 5.2 to measure the diversity. The metric we have chosen is also highlighted in the work by Zakaria et al. emphasizing its relevance and validity in evaluating procedural content generation [57]. In this analysis, we compute the average similarity score from each level to every other generated level.

Using the Hamming distance-based metric to evaluate our experiment has multiple advantages. First is its computational efficiency since we compare each level against all other generated levels. Second, its intuitive nature offers a clear and understandable measure of dissimilarity. We defined Diversity Score by the following equation:

$$D(\mathbf{l}gen, \mathbf{L}gen) = \frac{1}{|\mathbf{L}gen| - 1} \sum\nolimits_{\mathbf{l}'gen \in \mathbf{L}gen \ \mathbf{l}'gen \neq \mathbf{l}gen} \frac{d_H(\mathbf{l}gen, \mathbf{l}'gen)}{|\mathbf{l}_{gen}|} \tag{11}$$

In this equation, $\mathbf{l}gen$ is a level that has been procedurally generated, and $\mathbf{L}gen$ represents the set of all generated levels. $d_H(\mathbf{l}gen, \mathbf{l}'gen)$ is the Hamming distance between the generated level and another generated level. The sum of all these Hamming distances, each normalized by the size of the generated level $|\mathbf{l}gen|$, is then divided by the total number of generated levels minus one ($|\mathbf{L}gen| - 1$).

This Diversity Score signifies how different a generated level is from all other generated levels. A lower Diversity Score indicates that the generated level is very similar to other levels, and a higher Diversity Score denotes that the level is unique or different from other generated levels.

It is important to note that when calculating the Diversity Score, the level being evaluated is not included in the set of levels it is compared against (hence $\mathbf{l}'gen \neq \mathbf{l}gen$ in the sum). This is to ensure that each level is compared only against other, different levels.

Table 8 presents the diversity scores for generated Sokoban and Zelda levels using GAN, VAE, and VAEGAN models trained with baseline GVGAI dataset, extended dataset, and bootstrapping enabled.

Upon examination of the results from the baseline training data in the table, we observe that the VAE model has an average diversity score of 0.18, followed by VAEGAN with an average score of 0.17, and GAN with an average score of 0.12. The VAE model also exhibits the smallest standard deviation of 0.03, producing consistently diverse levels, whereas the GAN model has the largest standard deviation of 0.04, indicating a broader range of diversity.

Considering the results generated by models trained with the extended dataset levels, all three models exhibit higher diversity scores than when trained with the baseline data. Both VAE and VAEGAN have identical average scores of 0.20 and standard deviations of 0.03, marginally surpassing the GAN model with an average score of 0.18 and a standard deviation of 0.04.

For the bootstrapping training data, the VAE and VAEGAN models have identical average diversity scores of 0.17 and standard deviations of 0.03. The GAN model has a lower average diversity score of 0.13 but maintains a standard deviation of 0.04, consistent with the baseline dataset.

Table 8: Diversity scores of generated levels for Sokoban-Zelda blend.

Data	Model	Min	10%	25%	75%	90%	Max	Avg	Stdev
Baseline Dataset	GAN	0.05	0.07	0.10	0.19	0.20	0.21	0.12	0.04
	VAE	0.08	0.11	0.13	0.21	0.22	0.22	0.18	0.03
	VAEGAN	0.08	0.10	0.12	0.20	0.21	0.22	0.17	0.03
Extended Dataset	GAN	0.12	0.14	0.16	0.21	0.22	0.23	0.18	0.04
	VAE	0.14	0.16	0.18	0.22	0.23	0.23	0.20	0.03
	VAEGAN	0.14	0.16	0.18	0.22	0.23	0.23	0.20	0.03
Bootstrapping	GAN	0.07	0.09	0.11	0.20	0.21	0.21	0.13	0.04
	VAE	0.11	0.13	0.15	0.21	0.22	0.23	0.17	0.03
	VAEGAN	0.12	0.14	0.16	0.21	0.22	0.23	0.17	0.03

Table 9 outlines the diversity scores for generated Dungeon and Pacman levels across GAN, VAE, and VAEGAN models trained on baseline GVGAI dataset, on the extended dataset, and when bootstrapping enabled.

From the baseline data in Table 9, it is clear that VAEGAN has the highest average score of 0.32 with the lowest standard deviation of 0.04, demonstrating the most consistency. VAE follows with an average score of 0.30 and a standard deviation of 0.05. GAN trails the two with the lowest average score of 0.26 and the highest standard deviation of 0.06, indicating the most variability among the models.

Looking at the bootstrapping model, all models exhibit an increase in their scores relative to their baseline performances. GAN's average score increased to 0.28, VAE's to 0.31, while VAEGAN maintains the top position with an average score of 0.32. The standard deviations are 0.05 for GAN, 0.04 for VAE, and 0.04 for VAEGAN. The difference in scores between GAN and the other two models is less pronounced than in the baseline scenario.

Table 9: Diversity scores of generated levels for Dungeon-Pacman blend.

Data	Model	Min	10%	25%	75%	90%	Max	Avg	Stdev
Baseline Dataset	GAN	0.17	0.18	0.20	0.33	0.36	0.37	0.26	0.06
	VAE	0.21	0.23	0.26	0.36	0.37	0.38	0.30	0.05
	VAEGAN	0.24	0.26	0.28	0.37	0.37	0.38	0.32	0.04
Bootstrapping	GAN	0.20	0.21	0.24	0.35	0.37	0.37	0.28	0.05
	VAE	0.23	0.24	0.28	0.37	0.37	0.38	0.31	0.04
	VAEGAN	0.25	0.26	0.29	0.37	0.38	0.38	0.32	0.04

5.5 Comparison of Models

The 3D point cloud visualizations in Figures 18, 19, and 20 are presented to provide a comparison between different models. In the baseline dataset, as seen in Figure 18, the GAN model exhibits the lowest average novelty score, whereas the VAE and VAEGAN models present higher novelty. The extended dataset, visualized in Figure 19, indicates an overall increase in novelty scores, especially for the VAE model, which has the highest average score. With bootstrapping, as visualized in Figure 20, the novelty scores align closely with the baseline, with VAE and VAEGAN models outperforming GAN.

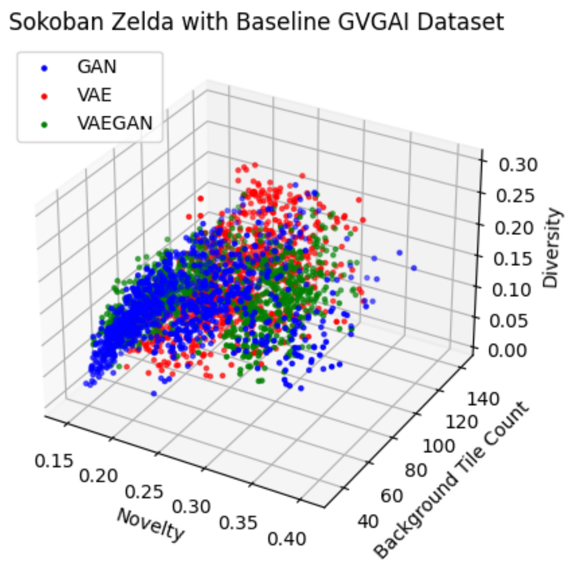


Figure 18: Visualization of Sokoban Zelda Blend Levels' Metrics in 3D Point Cloud, Generated Models Trained with the Baseline GVGAI Dataset.

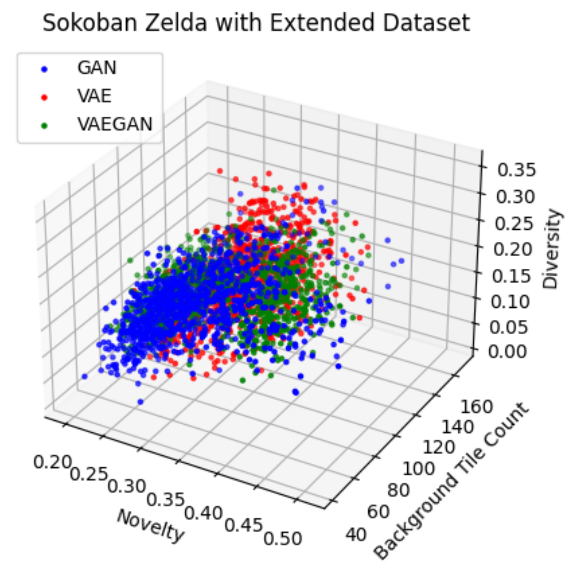


Figure 19: Visualization of Sokoban Zelda Blend Levels' Metrics in 3D Point Cloud, Generated Models Trained with the Extended Dataset.

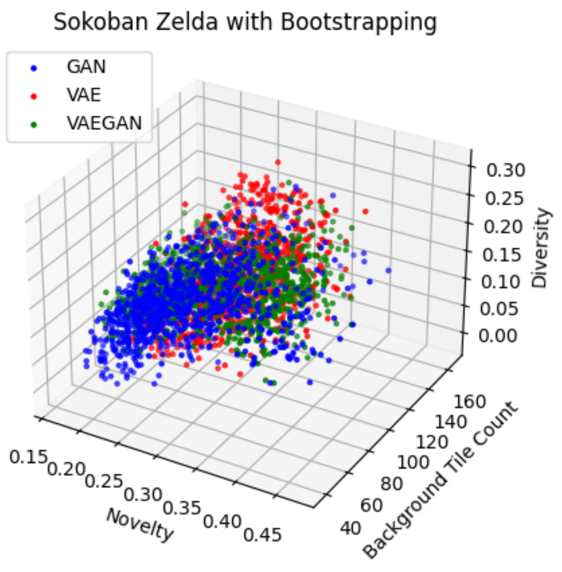


Figure 20: Visualization of Sokoban Zelda Blend Levels' Metrics in 3D Point Cloud, Generated Models Trained with Bootstrapping.

For the baseline dataset, the VAEGAN model tends to produce levels with higher wall tile counts, signifying denser levels, while the GAN model offers a more balanced distribution of wall and background tiles. In the extended dataset, the models exhibit similar trends in wall tile counts, but the GAN model shows the highest average number of background tiles. In the case of bootstrapping, the VAEGAN model continues to favor denser levels, while the GAN model maintains a balanced structure.

Examining the diversity scores, the baseline dataset reveals that the VAE model exhibits higher average diversity, followed closely by VAEGAN. In the extended dataset, the diversity scores improve, with VAE and VAEGAN displaying similar averages. The bootstrapping data shows that the VAE model sustains its lead, and the VAEGAN model exhibits increased diversity scores.

The 3D point cloud visualizations for Dungeon and Pacman blend levels are presented in Figures 21 and 22 to provide comparisons between different models trained with the baseline GVGAI dataset and bootstrapping.

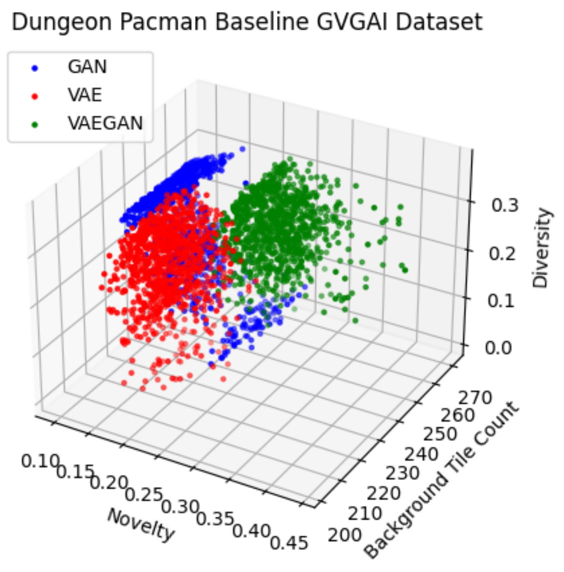


Figure 21: Visualization of Dungeon Pacman Blend Levels' Metrics in 3D Point Cloud, Generated Models Trained with Baseline GVGAI Dataset.

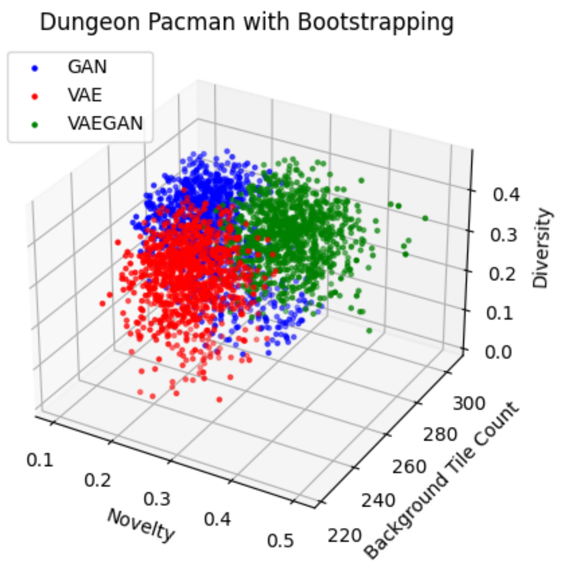


Figure 22: Visualization of Dungeon Pacman Blend Levels' Metrics in 3D Point Cloud, Generated Models Trained with Bootstrapping.

In terms of novelty scores, as depicted in Table 5, the baseline dataset reveals that VAE and VAEGAN models create almost identical scores while GAN's average is lower. The bootstrapping dataset exhibits higher novelty scores, with VAE and VAEGAN outpacing GAN's average. This indicates that VAE and VAEGAN models consistently produce more novel levels, while GAN tends to be more similar to its training data.

Tile densities, as observed in Table 7, show that in the baseline data, GAN exhibits a tight distribution, suggesting consistency, whereas VAEGAN and VAE display more varied level designs. In the bootstrapping data, the differences begin to converge, but VAEGAN and VAE continue to produce levels with greater variability in design, while GAN remains more consistent.

The diversity scores, outlined in Table 9, reveal that VAEGAN has the highest average score in the baseline data, surpassing VAE and GAN. In the bootstrapping model, all models see a slight increase in their scores relative to their baseline performances.

5.6 Analysis of Levels Generated by RL-Based Latent Vector Optimization

In a separate series of experiments, we introduced Reinforcement Learning (RL) into our methodology as an alternative to the A* solver in the CMA-ES fitness function latent vector optimization. Training of the RL agent was performed independently for each game. Thus, the RL agent was trained separately for the Dungeon and Pacman games. Then, we expected this RL agent to play a blend of both games. We conducted experiments using the baseline GVGAI dataset, generating 1,000 levels per blend using the three generative models - GAN, VAE, and VAEGAN. As before, the evaluation metrics utilized were Novelty and Diversity.

Table 10 presents the Novelty scores of the generated levels for Dungeon and Pacman using RL-based Latent Vector Optimization. It is observed from the results that the RL agent generated levels with considerably reduced novelty compared to the levels from other methodologies.

Table 10: Novelty scores of generated levels for Dungeon and Pacman using RL-based Latent Vector Optimization.

Data	**Model**	**Min**	**10%**	**25%**	**75%**	**90%**	**Max**	**Avg**	**Stdev**
Baseline Dataset	GAN	0.05	0.07	0.08	0.12	0.14	0.15	0.10	0.03
	VAE	0.06	0.08	0.09	0.12	0.13	0.13	0.10	0.02
	VAEGAN	0.06	0.07	0.09	0.13	0.13	0.14	0.10	0.02

Lastly, the Diversity scores, which are indicative of the variety in the generated levels, are presented in Table 11. The results suggest a significant decrease in the diversity score when using the RL-based fitness function, indicating that the generated levels tend to be more similar to one another.

Table 11: Diversity scores of generated levels for Dungeon and Pacman using RL-based Latent Vector Optimization.

Training Data	**Model**	**Min**	**10%**	**90%**	**Max**	**Avg**	**Stdev**
Baseline Dataset	GAN	0.01	0.02	0.05	0.07	0.04	0.02
	VAE	0.01	0.03	0.06	0.08	0.05	0.02
	VAEGAN	0.01	0.03	0.07	0.09	0.05	0.02

The results for the RL agent were not entirely satisfactory. The RL agent generated levels exhibiting less novelty and diversity than those produced by the preceding two-step fitness function. The data in Tables 10 and 11 underscore this observation, highlighting that the RL agent tends to generate game levels highly similar to the dataset levels. This suggests that RL-based optimization might be producing outputs with high similarity, which can be an area of further investigation.

CHAPTER 6

DISCUSSION

The research presented in this thesis focuses on the novel method of blending games, leveraging the GVGAI framework to integrate game descriptions and level representations. Our ultimate aim is to create a complete methodology for blending game levels. Blending game descriptions includes blending game descriptions which are the combination of game objects, interaction rules between objects, and finishing rules. For a blended-level generation, we experimented with three generative networks VAE, GAN, and a hybrid model named VAEGAN. Furthermore, to ensure the playability of the blended games, the CMA-ES was incorporated into the generator networks to optimize the latent vectors of GAN, VAE, and VAEGAN models; within the heuristic scoring function of the CMA-ES optimization, we used A* agents to generate playable levels.

Regarding game description blending, it is worth noting that our approach has a similar objective to Gow and Corneli's approach. Gow and Corneli's process involves generalizing the games to a common concept, blending them according to these concepts, and evaluating the blend. They also include a step for weakening the game, where certain elements are modified or removed to ensure game coherence, and a step for running the blend, which involves adjusting the game to take advantage of the blended elements. In contrast, our methodology directly integrates elements from the two games by assessing compatibility, identifying shared elements, merging game objects, and managing potential conflicts in symbolic representations, interactions, and termination rules. This direct integration of elements eliminates the need for the generalization step and the weakening phase of Gow and Corneli's methodology. However, our process also includes an implicit evaluation component within each step. Our method represents a significant step forward in automating the process of blending game descriptions while still maintaining the necessary human touch to manage contextual considerations. By providing a systematic and replicable approach, we anticipate this method will expand the possibilities of game blending.

In our blended game level generation experiments, we evaluated the effectiveness of GANs, VAEs, and VAEGANs in blending game levels, explicitly highlighting their capability to produce novel, complex, and diverse blended levels. We can summarize our findings as follows: In our research, we conducted an extensive analysis of the novelty of the generated levels. Initially, our focus was to understand how distinct the generated levels are when compared to the baseline training dataset. Using the Hamming distance-based novelty score, our findings suggested that VAEs and VAEGANs had a slight edge over GANs in terms of creating unique levels. For instance, in the Dungeon and Pacman blend, all three models displayed nearly identical novelty results. However, for the Zelda and Sokoban blend, GANs lagged behind, producing levels that closely resembled the training set. This pointed towards a potential drawback of the GAN model when dealing with datasets that are smaller and simpler in structure.

However, upon expanding our study to include levels generated using the extended dataset and using Bootstrapping in training, we observed some intriguing patterns. For levels generated from the extended dataset, all three models—GAN, VAE, and VAEGAN—exhibited higher novelty scores compared to those generated using the Baseline dataset. This indicates that when trained on more levels, the models seem to produce more diverse and unique content. The VAE model showcased the highest average novelty score, followed closely by VAEGAN, with GAN trailing behind.

Moreover, when we considered levels generated using the Bootstrapping dataset, the novelty scores were closely aligned with those from the Baseline dataset. Both VAE and VAEGAN models consistently outperformed the GAN model. Notably, GAN, across all datasets, demonstrated a more extensive range in its scores, suggesting a broader spectrum of output that could either be closely related or distinct from the training levels. The second observation involved analyzing the tile density within the game levels, a key metric for understanding a level's complexity. When examining the results, we found that VAEGANs consistently outperformed both VAEs and GANs in terms of wall tile density across all datasets: baseline, extended, and bootstrapping. Specifically, VAEGANs tended to produce levels with more wall tiles and fewer background tiles, implying denser and more intricate levels. VAEs, interestingly, showed performance metrics in between that of GANs and VAEGANs. This was particularly evident in the Sokoban and Zelda experiments. From this, we can infer that for smaller level sizes, VAEs exhibit improved performance.

Looking at the extended levels, a similar trend persisted, but the differences were less pronounced. Each model, be it VAEGAN, VAE, or GAN, displayed only slight variances in wall and background tile density. This suggests that while there's a difference in the levels produced by each generative model when more data is available, these differences become subtler.

Lastly, in the bootstrapping dataset, VAEGAN continued to produce denser levels, whereas the VAE and GAN models presented levels with more open spaces or background tiles. Of note, the GAN model seemed to strike a balance between wall and background tile distribution, possibly offering a more moderate gameplay experience.

To summarize, VAEGAN consistently leans towards generating denser, potentially more challenging levels across datasets. Meanwhile, GANs often produce levels with a balance of open space and obstacles, and the performance of VAEs seems to be contingent on level size. The final observation assessed the diversity among the generated levels. We introduced a diversity score based on Hamming distance. For games like Dungeon and Pacman, the VAEGAN model, when using an extended dataset, did better. It made more varied levels than before. The VAE model also made more varied levels when we used bootstrapped data. For the Sokoban and Zelda games, things were a bit different. All the models made levels that were quite similar in variety. The VAE model made slightly more varied levels when we used extended data. But the VAEGAN model was almost as good when we used bootstrapped data.

When we trained the models with more levels, they made the most varied levels. This shows that using more level data in training helps the models make better levels. Our new tests tell us two things. First, for small game spaces, the VAE model is better. For bigger spaces, the VAEGAN model is better. Second, the kind of training data we use matters a lot. Using different training data can help the models make better game levels. Common points of all three experiments are that VAEGANs outperformed the GANs and VAEs in all blending Dungeon and Zelda experiments demonstrating their capability to generate diverse and complex levels. In contrast, GAN models produce samples with less diversity and complexity, which reveals potential discriminator overfitting problems with GAN. GANs are known to be sensitive to discriminator overfitting, which can happen when the dataset is not large enough. Given that our dataset is limited to GVGAI samples, discriminator overfitting likely occurred in our experiments with Baseline GVGAI dataset. Discriminators can memorize all of the datasets, which results in a generator producing a limited diversity of samples called mode collapse. Performance of GAN models increased more than other models when we used Extended dataset, which also confirms discriminator overfitting with small data sizes. Finally, we observed that within smaller environments VAEs perform greater, even exceeding VAEGANs performance in some cases. In our experiments with integrating Reinforcement Learning Agents into the CMA-ES fitness function yielded different results than expected. The performance decline when using RL might be attributed to several factors: Firstly, our assumption that agents trained separately on individual games would perform well on a blended version may only partially be correct. Secondly, further optimization of the RL agent's engineering could have improved outcomes. Finally, the comparison between RL and A* path-finding algorithms may need to be revised for the maze-like games we analyzed, where deterministic solutions give A* an advantage. These insights underline the necessity of considering the game's nature, training methodologies, and the chosen algorithms' nuances when optimizing game level blending.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This thesis introduces a novel method for blending games, focusing on blending GVGAI game descriptions and level representations. The main objective of our research is to create blended game levels that are novel, diverse, and playable. Our approach consists of blending game descriptions in a structured way, using generative models to create blended level representations, and finally optimizing generator networks of GAN, VAE, and VAEGAN models to guarantee the playability of level creations.

With rigorous experimentation, we have shown that our proposed methodology can generate blended game levels that are playable. We also discussed that our approach for game level blending has several advantages over previously introduced methods. Compared to previous research in game blending, we showed a complete set of methods required for blending games: blending levels and game descriptions that define game objects and interactions between objects. Thus, in our study we achieved to have playable novel games that carry characteristics of both of their ancestors. Furthermore, these blended games and levels can work within the GVGAI Java simulator.

Our empirical observations show the efficacy of VAEGANs in outperforming VAEs and GANs in creating complex levels with greater diversity, especially in blending Dungeon and Pacman game levels which have larger map sizes compared to Sokoban and Zelda combination. On the other hand, VAEs exhibited to be a good alternative when the size of the game levels is small, which we observe from blending Sokoban and Zelda. Although there are several examples of using GANs in PCG literature, in this research, GANs show the poorest performance among three generative networks, and we attribute these results to our limited data size. As per the literature, GANs require more data compared to VAEs.

Our experiment with Reinforcement Learning (RL) revealed areas for improvement and further exploration. The integration of RL into our methodology led to certain unexpected outcomes, such as performance drop, attributed to several factors, like our assumptions about RL agents' efficacy in playing blended games, and potential improvements in the RL agent's engineering.

While our research has demonstrated promising outcomes, it also underscores potential avenues for future exploration. Specifically, refining the methodology to blend a broader spectrum of games, potentially extending beyond the GVGAI framework, is a plausible future direction. In addition, advanced metrics that include gameplay dynamics and physical layout could contribute to more detailed measurements of novelty and diversity. Further, incorporating reinforcement learning agents for game exploration could result in more balanced, engaging, and dynamic game levels.

Another exciting research prospect lies in automating the game blending process further. With the advent of Large Language Models (LLMs), we envision the possibility of automating the blending of game rules and interactions. Due to their capacity for contextual understanding, LLMs can replace manual work in blending game descriptions.

In conclusion, our research is an essential step in game blending, providing an alternative and structured way to blend GVGAI game descriptions and illustrating the potential blending level representations within the GVGAI framework using the VAEGAN model.

REFERENCES

[1] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[2] R. Van Der Linden, R. Lopes, and R. Bidarra, "Procedural generation of dungeons," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78–89, 2013.

[3] Y. I. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 301–308, ACM, 2001.

[4] W. M. Reis, L. H. S. Lelis, and Y. K. Gal, "Human computation for procedural content generation in platform games," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 99–106, IEEE, Aug 2015.

[5] N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for super mario bros using grammatical evolution," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 304–311, IEEE, 2012.

[6] M. J. Nelson, J. Togelius, C. Browne, and M. Cook, "Rules and mechanics," *Procedural Content Generation in Games*, pp. 99–121, 2016.

[7] J. Togelius, N. Shaker, and M. J. Nelson, "The search-based approach," *Computational Synthesis and Creative Systems, Cham, Springer*, pp. 17–30, 2016.

[8] J. Togelius, M. Preuss, and G. N. Yannakakis, "Towards multiobjective procedural map generation," in *Proceedings of the 2010 workshop on procedural content generation in games*, p. 3, ACM, 2010.

[9] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *European Conference on the Applications of Evolutionary Computation*, pp. 131–140, Springer, 2010.

[10] C. B. Browne, *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

[11] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[12] A. J. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," in *DiGRA/FDG*, 2016.

[13] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

[14] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, "Predicting resource locations in game maps using deep convolutional neural networks," in *The Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AAAI, 2016.

[15] J. Gow and J. Corneli, "Towards generating novel games using conceptual blending," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 11, pp. 15–21, 2015.

[16] M. Guzdial and M. Riedl, "Learning to blend computer game levels," *arXiv preprint arXiv:1603.02738*, 2016.

[17] A. Sarkar and S. Cooper, "Blending levels from different games using lstms.," in *AIIDE Workshops*, 2018.

[18] A. Sarkar and S. Cooper, "Dungeon and platformer level blending and generation using conditional vaes," in *2021 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2021.

[19] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

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

[23] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine, "Variational discriminator bottleneck: Improving imitation learning, inverse RL, and GANs by constraining information flow," in *International Conference on Learning Representations*, 2019.

[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[25] X. Yu, X. Zhang, Y. Cao, and M. Xia, "Vaegan: A collaborative filtering framework based on adversarial variational autoencoders.," in *IJCAI*, pp. 4206–4212, 2019.

[26] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[27] T. Schaul, "A video game description language for model-based or interactive learning," in *IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, IEEE, 2013.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, p. 1, 2013.

[30] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, Neural Information Processing Systems Foundation, 2014.

[32] S. Snodgrass and S. Ontañón, "Learning to generate video game maps using markov models," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 4, pp. 410–422, 2017.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, Neural Information Processing Systems Foundation, 2012.

[34] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[35] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[36] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[37] P. Bontrager, A. Roy, J. Togelius, N. Memon, and A. Ross, "Deepmasterprints: Generating masterprints for dictionary attacks via latent variable evolution," in *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pp. 1–9, IEEE, 2019.

[38] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 221–228, ACM, 2018.

[39] E. Giacomello, P. L. Lanzi, and D. Loiacono, "Doom level generation using generative adversarial networks," in *IEEE Games, Entertainment, Media Conference (GEM)*, IEEE, 2018.

[40] J. Gutierrez and J. Schrum, “Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda,” *arXiv preprint arXiv:2001.05065v1*, 2020.

[41] J. Schrum, V. Volz, J. Liu, S. M. Lucas, A. Smith, and S. Risi, “Interactive evolution and exploration within latent level-design space of generative adversarial networks,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ACM, 2020.

[42] S. Thakkar, C. Cao, L. Wang, T. J. Choi, and J. Togelius, “Autoencoder and evolutionary algorithm for level generation in lode runner,” in *IEEE Conference on Games*, 2019.

[43] Z. Yang, A. Sarkar, and S. Cooper, “Game level clustering and generation using gaussian mixture vaes,” in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.

[44] A. Sarkar, Z. Yang, and S. Cooper, “Controllable level blending between games using variational autoencoders,” in *EXAG Workshop*, 2019.

[45] A. Sarkar, A. Summerville, S. Snodgrass, G. Bentley, and J. Osborn, “Exploring level blending across platformers via paths and affordances,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, pp. 280–286, 2020.

[46] A. Sarkar and S. Cooper, “Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder,” in *Proceedings of the 16th International Conference on the Foundations of Digital Games*, pp. 1–11, 2021.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[49] A. P. Badia, J. Modayil, C. Blundell, T. Schaul, D. Hassabis, and D. Silver, “Agent57: Outperforming the atari human benchmark,” *arXiv preprint arXiv:2003.13350*, 2020.

[50] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, “Deep reinforcement learning for general video game ai,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, IEEE, 2018.

[51] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, “Illuminating generalization in deep reinforcement learning through procedural level generation,” *arXiv preprint arXiv:1806.10729*, 2018.

[52] A. Guez, M. Mirza, K. Gregor, R. Kabra, S. Racaniere, T. Weber, D. Raposo, A. Santoro, L. Orseau, T. Eccles, G. Wayne, D. Silver, T. Lillicrap, and V. Valdes, “An investigation of model-free planning: boxoban levels.” https://github.com/deepmind/boxoban-levels/, 2018.

[53] 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.

[54] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[56] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," in *2020 IEEE Conference on Games (CoG)*, pp. 41–48, IEEE, 2020.

[57] Y. Zakaria, M. Fayek, and M. Hadhoud, "Procedural level generation for sokoban via deep learning: An experimental study," *IEEE Transactions on Games*, vol. 15, no. 1, pp. 108–120, 2022.

[58] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," *CoRR*, vol. abs/1910.01603, 2019.

[59] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

APPENDIX A

BASELINE GVGAI DATASET

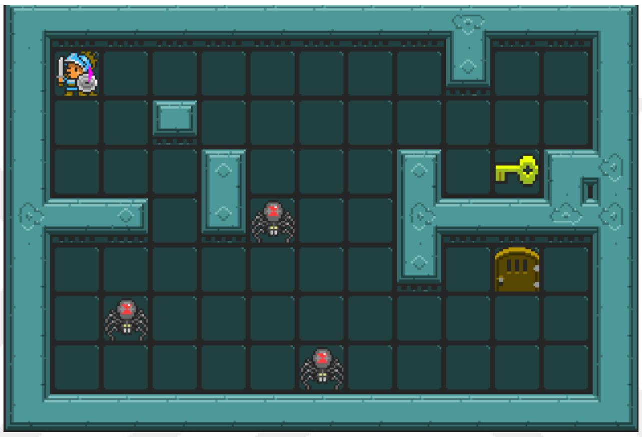


Figure A1: Baseline GVGAI Dataset Zelda Level 1.

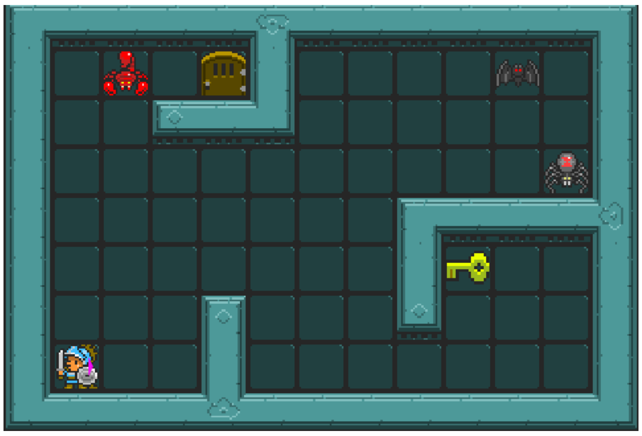


Figure A2: Baseline GVGAI Dataset Zelda Level 2.

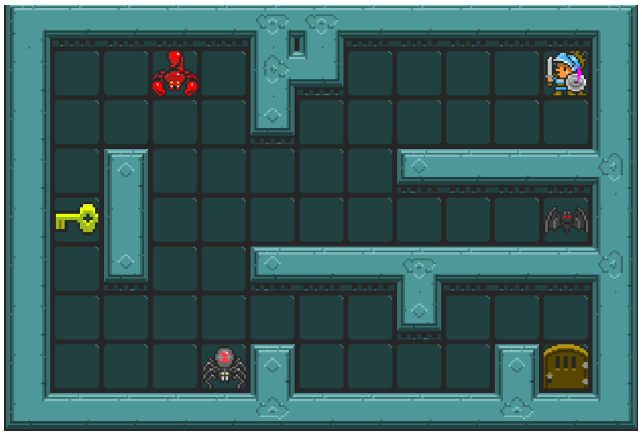


Figure A3: Baseline GVGAI Dataset Zelda Level 3.

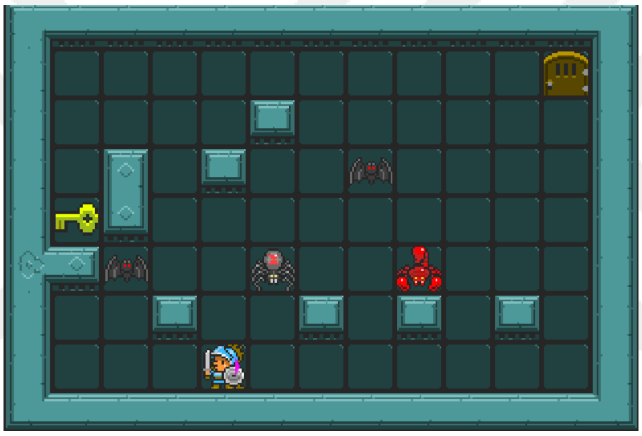


Figure A4: Baseline GVGAI Dataset Zelda Level 4.

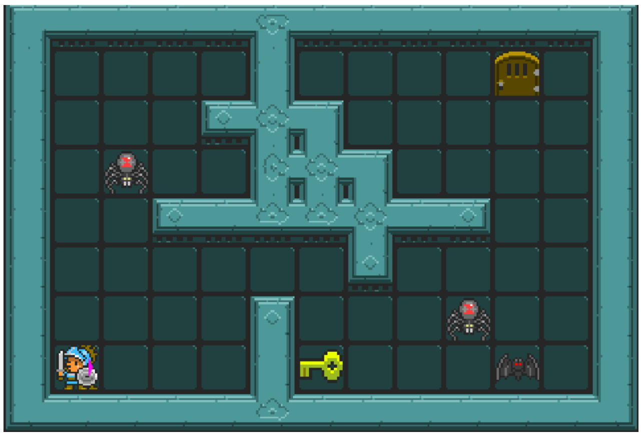


Figure A5: Baseline GVGAI Dataset Zelda Level 5.

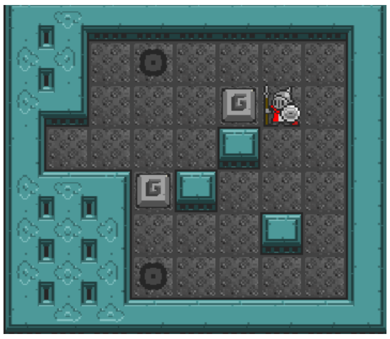


Figure A6: Baseline GVGAI Dataset Sokoban Level 1.

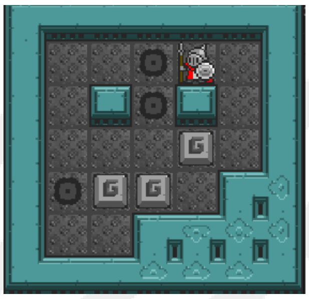


Figure A7: Baseline GVGAI Dataset Sokoban Level 2.

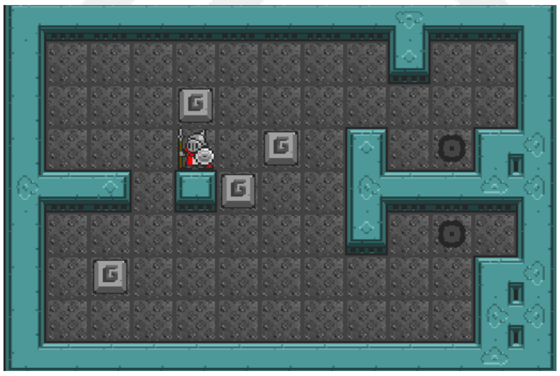


Figure A8: Baseline GVGAI Dataset Sokoban Level 3.

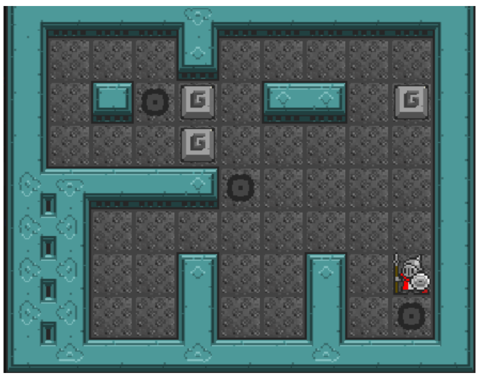


Figure A9: Baseline GVGAI Dataset Sokoban Level 4.

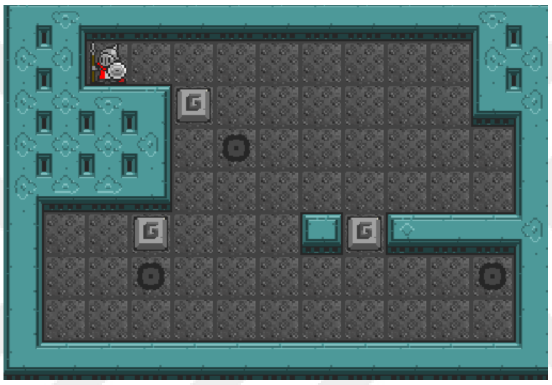


Figure A10: Baseline GVGAI Dataset Sokoban Level 5.

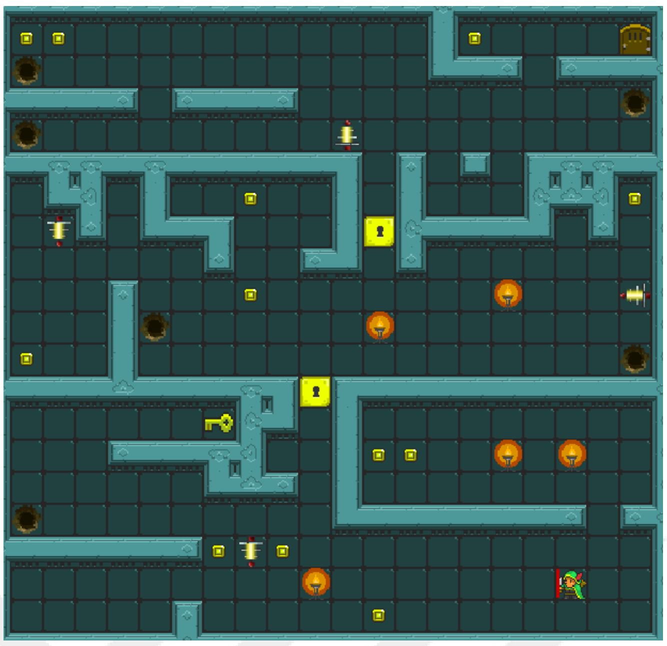


Figure A11: Baseline GVGAI Dataset Dungeon Level 1.

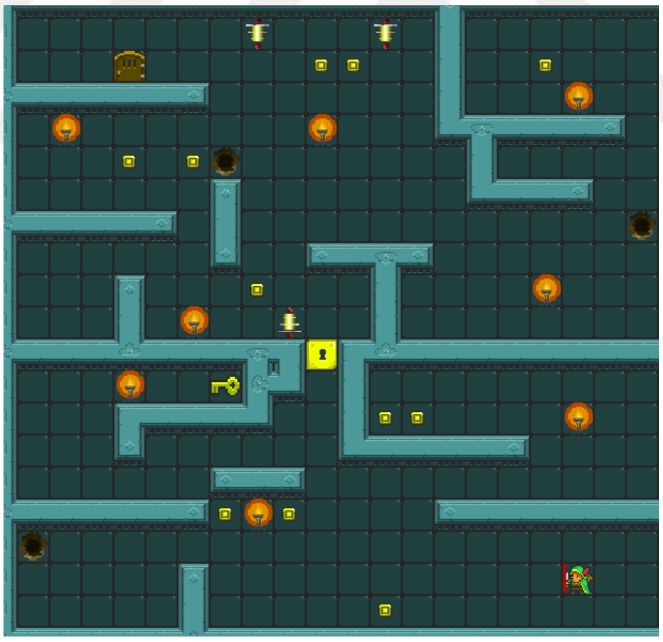


Figure A12: Baseline GVGAI Dataset Dungeon Level 2.

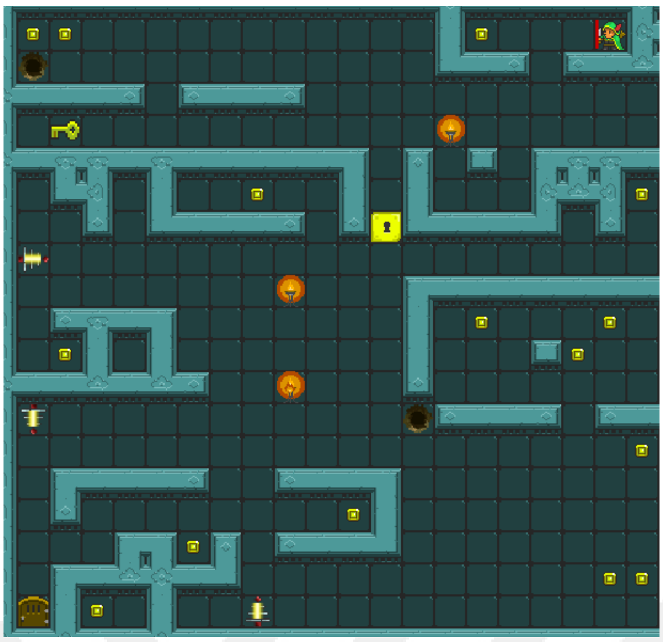


Figure A13: Baseline GVGAI Dataset Dungeon Level 3.

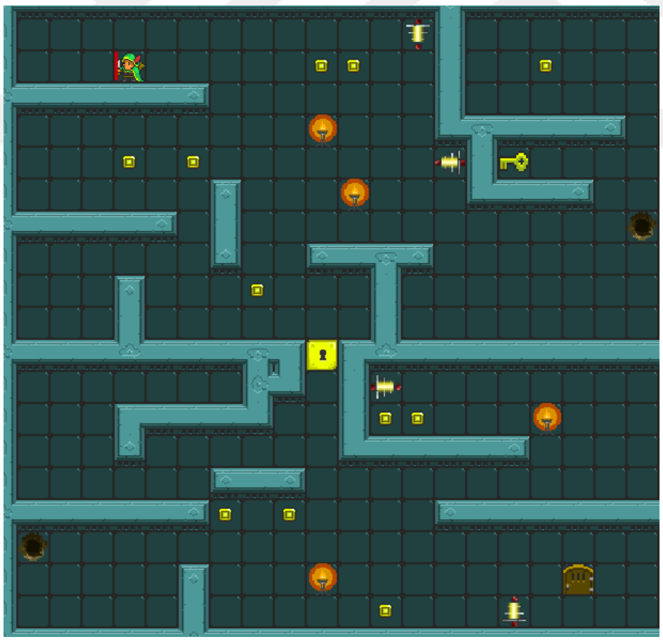


Figure A14: Baseline GVGAI Dataset Dungeon Level 4.

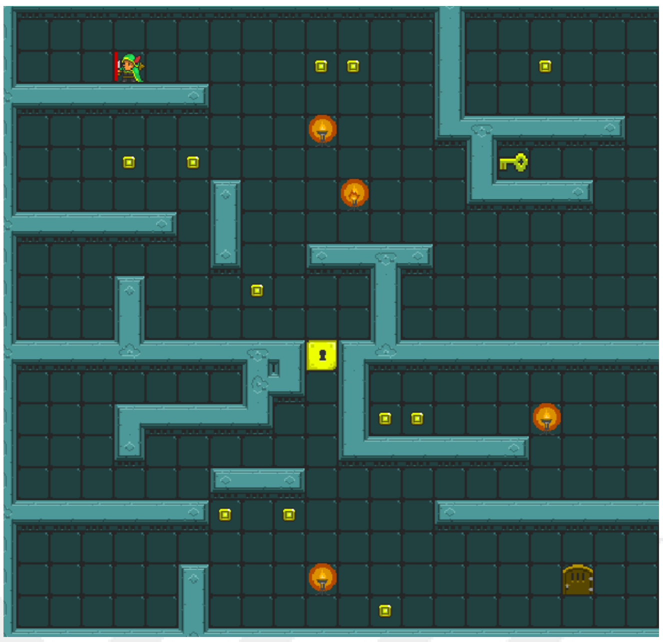


Figure A15: Baseline GVGAI Dataset Dungeon Level 5.

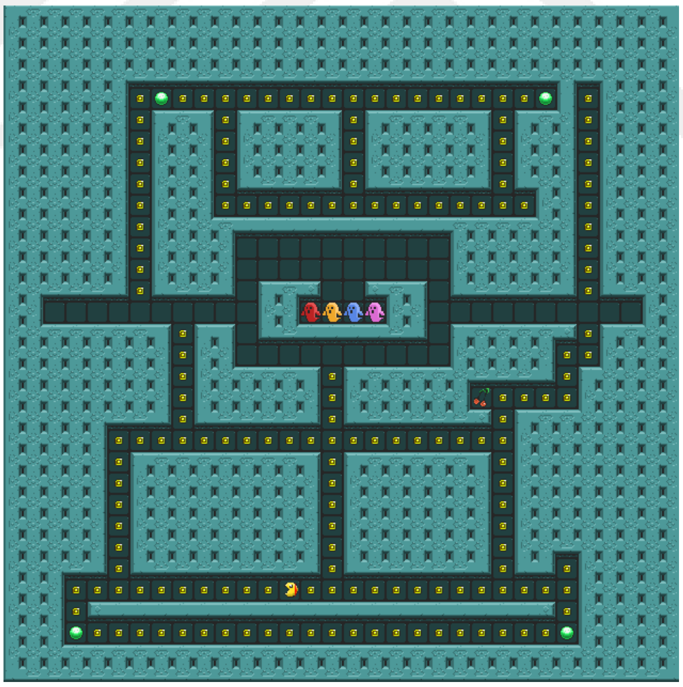


Figure A16: Baseline GVGAI Dataset Pacman Level 1.

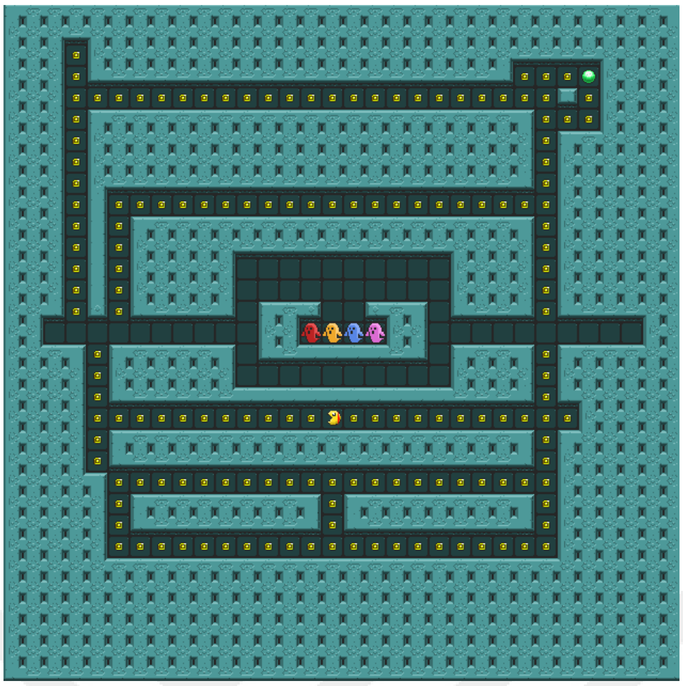


Figure A17: Baseline GVGAI Dataset Pacman Level 2.

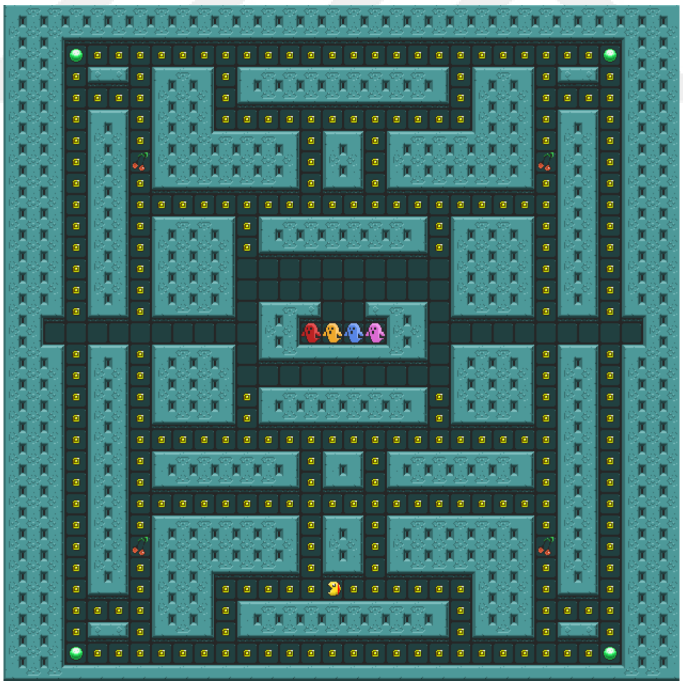


Figure A18: Baseline GVGAI Dataset Pacman Level 3.

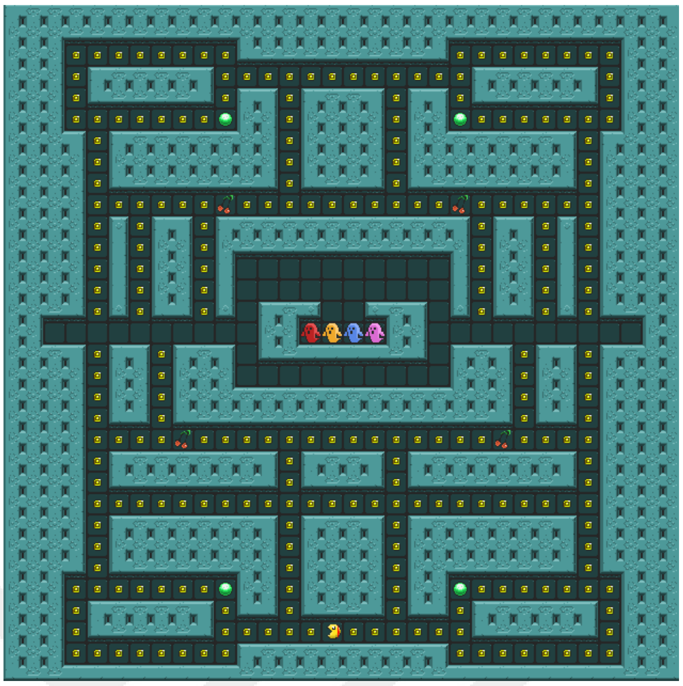


Figure A19: Baseline GVGAI Dataset Pacman Level 4.

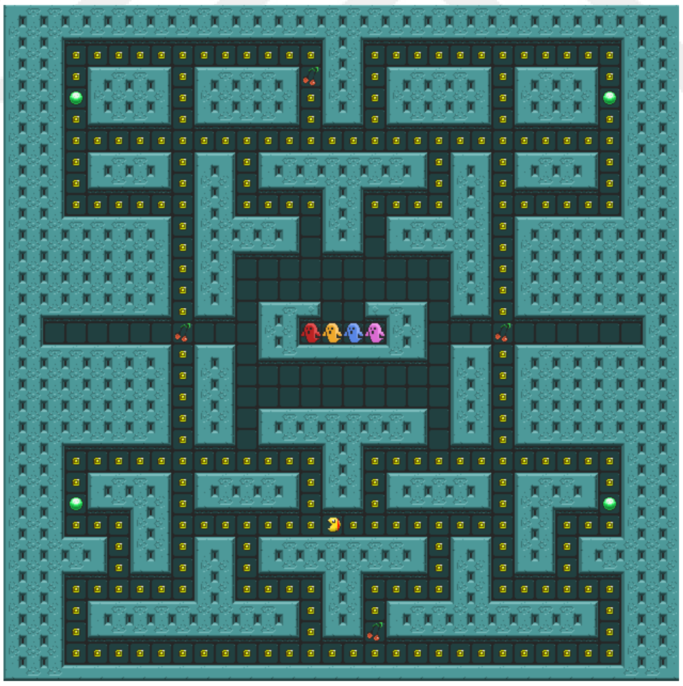


Figure A20: Baseline GVGAI Dataset Pacman Level 5.

APPENDIX B

GENERATIVE NETWORK PARAMETERS

Table B1: Model evaluation results for VAE trained on Baseline GVGAI Dataset for Dungeon Pacman Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	4000	0.998	0.31
64	64	3000	0.998	0.297
64	128	1000	0.998	0.289
32	64	5000	0.996	0.283
64	256	6000	0.982	0.277
64	64	7000	0.978	0.269
64	256	7000	0.966	0.263
64	64	6000	0.941	0.256
64	128	9000	0.939	0.248
64	128	7000	0.897	0.243
32	64	4000	0.893	0.238
64	64	5000	0.878	0.232
64	64	8000	0.809	0.225
64	256	5000	0.002	0.218

Table B2: Model evaluation results for VAEGAN trained on Baseline GVGAI Dataset for Dungeon Pacman Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	5000	1	0.35
64	128	8000	1	0.34
64	64	7000	1	0.331
64	64	9000	1	0.323
64	128	3000	1	0.317
64	128	2000	0.952	0.308
64	128	6000	0.912	0.301
64	128	1000	0.602	0.295
64	256	2000	0.52	0.288

Table B3: Model evaluation results for GAN trained on Baseline GVGAI Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
32	64	4000	0.769	0.181
64	128	2000	1	0.178
32	64	3000	1	0.168
32	64	2000	0.951	0.158
64	128	6000	0.841	0.148
32	64	7000	0.6	0.128
32	64	10000	0.567	0.118
64	256	5000	0.512	0.198

Table B4: Model evaluation results for VAE trained on Baseline GVGAI Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
32	64	4000	1	0.28
64	128	2000	0.999	0.262
64	256	3000	0.999	0.246
32	64	3000	0.998	0.23
64	64	2000	0.998	0.214
32	64	2000	0.996	0.198
32	64	5000	0.995	0.182
64	256	7000	0.487	0.166

Table B5: Model evaluation results for VAEGAN trained on Baseline GVGAI Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
32	64	7000	1	0.27
32	64	4000	1	0.26
64	256	3000	1	0.25
64	256	2000	1	0.24
64	256	4000	1	0.23
64	128	4000	1	0.22
64	128	6000	1	0.21
64	64	6000	1	0.20
64	128	7000	1	0.19
64	128	9000	1	0.18
32	64	1000	0.989	0.17
64	256	7000	0.512	0.16

Table B6: Model evaluation results for GAN trained on Extended Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	6000	0.941	0.231
64	128	3000	1	0.22
32	64	5000	1	0.218
64	256	3000	0.951	0.215
64	128	8000	0.841	0.201
32	64	8000	0.6	0.2
64	256	6000	0.567	0.198
64	256	7000	0.512	0.193

Table B7: Model evaluation results for VAE trained on Extended Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	4000	1	0.38
64	128	3000	0.999	0.362
64	256	4000	0.999	0.346
32	64	4000	0.998	0.33
64	64	3000	0.998	0.314
32	64	3000	0.996	0.298
64	256	8000	0.995	0.282
64	256	9000	0.487	0.266

Table B8: Model evaluation results for VAEGAN trained on Extended Dataset for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	8000	1	0.37
64	256	5000	1	0.36
64	256	4000	1	0.35
64	256	3000	1	0.34
64	128	5000	1	0.33
64	256	4000	1	0.32
64	128	7000	1	0.31
64	64	7000	1	0.30
64	64	8000	1	0.29
64	128	4000	1	0.28
64	64	10000	0.512	0.26

Table B9: Model evaluation results for GAN trained with Bootstrapping for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	6000	0.941	0.198
32	64	9000	0.919	0.186
64	128	4000	1	0.183
32	64	6000	1	0.173
32	64	5000	0.951	0.163
32	64	9000	0.6	0.133
32	64	10000	0.567	0.123
64	256	5000	0.512	0.103

Table B10: Model evaluation results for VAE trained with Bootstrapping for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	6000	0.999	0.312
32	64	8000	1	0.305
64	256	3000	0.999	0.291
32	64	9000	0.998	0.275
64	64	5000	0.998	0.259
32	64	6000	0.996	0.243
32	64	5000	0.995	0.227
64	256	6000	0.487	0.211

Table B11: Model evaluation results for VAEGAN trained with Bootstrapping for Sokoban Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	256	3000	1	0.308
64	128	7000	1	0.295
64	256	4000	1	0.283
32	64	7000	1	0.277
32	64	4000	1	0.263
64	256	2000	1	0.259
64	128	4000	1	0.241
64	128	6000	1	0.235
64	64	6000	1	0.226
64	128	7000	1	0.216
32	64	10000	0.989	0.201
64	256	7000	0.512	0.193

Table B12: Model evaluation results for GAN trained with Bootstrapping for Dungeon Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	6000	0.955	0.208
32	64	8000	0.932	0.196
64	128	4000	1	0.189
32	64	6000	1	0.179
32	64	5000	0.958	0.169
64	128	9000	0.621	0.139
32	64	90000	0.582	0.129
64	256	6000	0.527	0.109

Table B13: Model evaluation results for VAE trained with Bootstrapping for Dungeon Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	7000	1	0.322
64	128	6000	1	0.315
64	256	3000	0.999	0.298
32	64	9000	0.998	0.285
64	64	5000	0.998	0.269
32	64	6000	0.997	0.253
32	64	5000	0.995	0.237
64	256	6000	0.493	0.221

Table B14: Model evaluation results for VAEGAN trained with Bootstrapping for Dungeon Zelda Blend.

ngf	nz	epoch	uniques ratio	playability ratio
64	128	8000	1	0.318
64	128	7000	1	0.305
64	256	4000	1	0.293
32	64	7000	1	0.287
32	64	4000	1	0.273
64	256	2000	1	0.269
64	128	6000	1	0.251
64	128	5000	1	0.245
64	64	6000	1	0.236
32	64	10000	0.992	0.211
64	256	7000	0.517	0.203

Table B15: General Parameters Used in All Models.

Parameter	**Value**
batchSize	8
lrD (learning rate for the critic)	0.00005
lrG (learning rate for generator)	0.00005
beta1 (beta1 for adam)	0.5
cuda (enables cuda)	True if available
ngpu (number of GPUs to use)	1
Diters (number of D iters per G iter)	5
n_extra_layers (extra layers on gen and disc)	1

APPENDIX C

SAMPLE OUTPUTS OF MERGED LEVELS

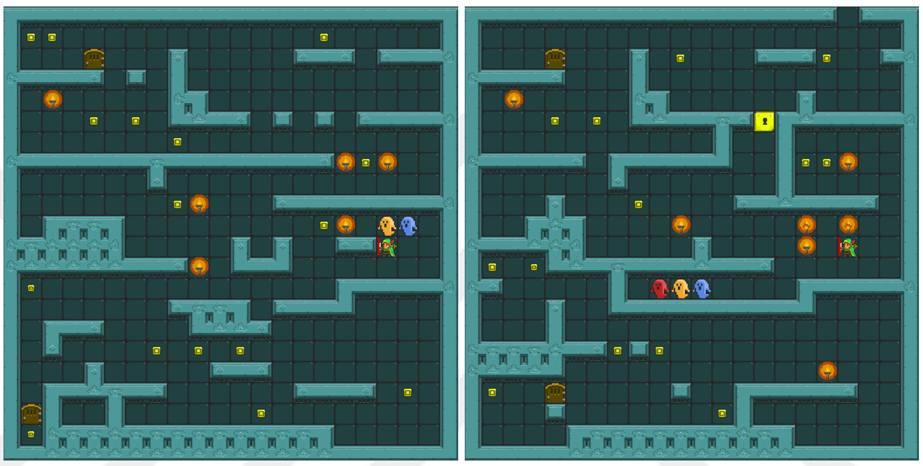


Figure C1: Examples of Dungeon and Pacman Blend Generated by GAN trained with Baseline GVGAI Dataset.

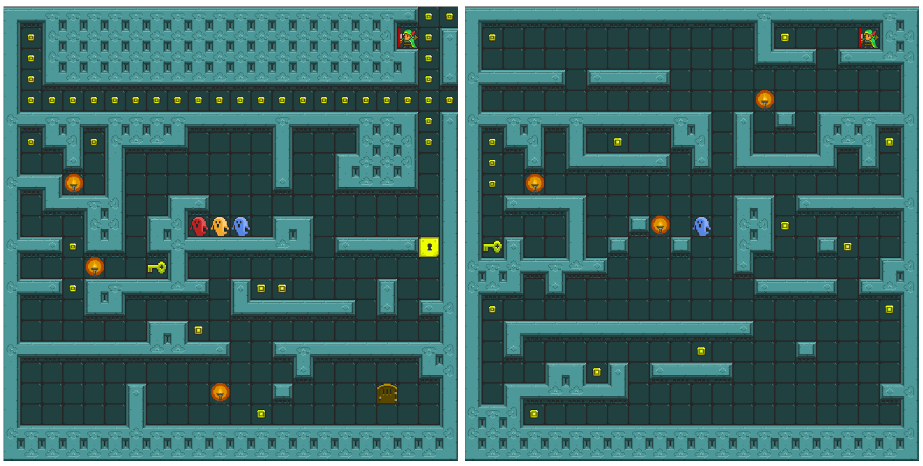


Figure C2: Examples of Dungeon and Pacman Blend Generated by VAE trained with Baseline GVGAI Dataset.

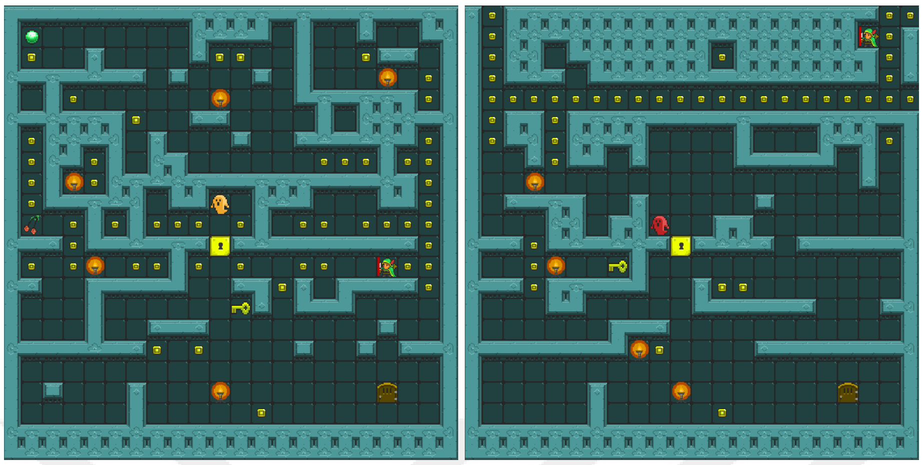


Figure C3: Examples of Dungeon and Pacman Blend Generated by VAEGAN trained with Baseline GVGAI Dataset.

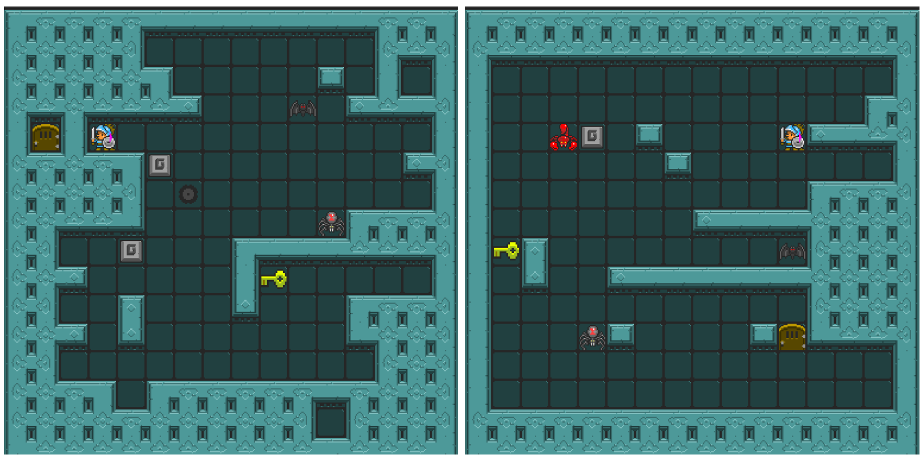


Figure C4: Examples of Zelda and Sokoban Blend Generated by GAN trained with Baseline GVGAI Dataset.

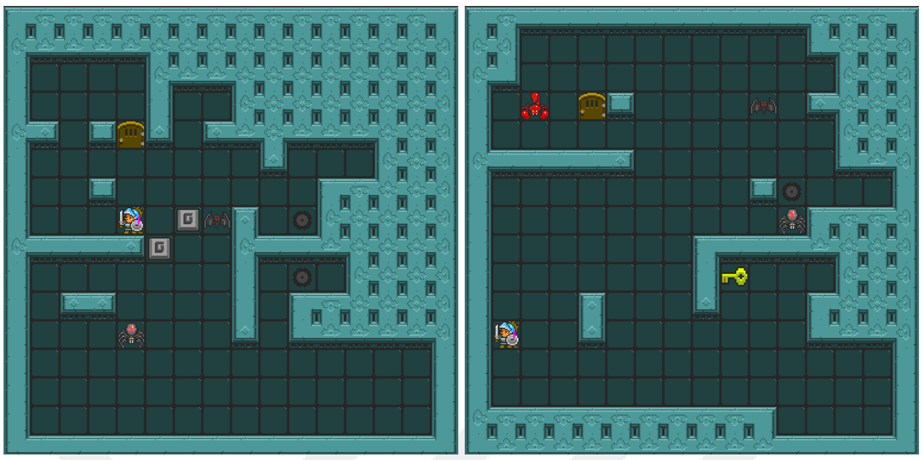


Figure C5: Examples of Zelda and Sokoban Blend Blend Generated by VAE trained with Baseline GVGAI Dataset.

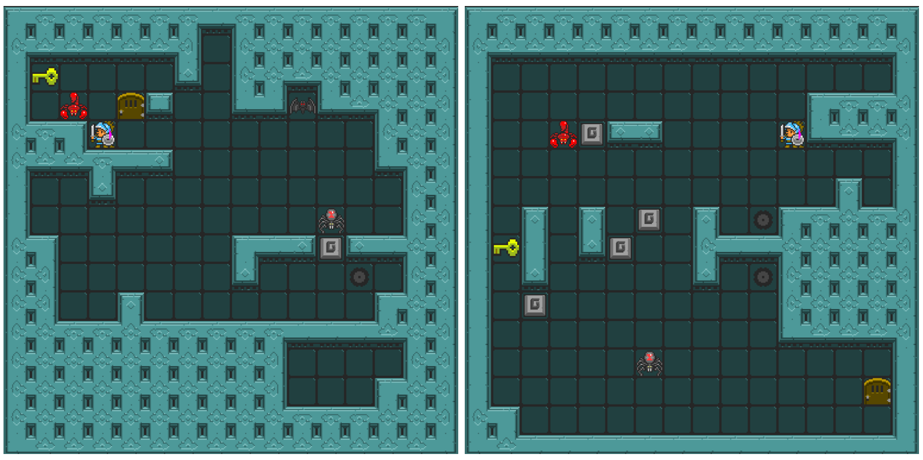


Figure C6: Examples of Zelda and Sokoban Blend Generated by VAEGAN trained with Baseline GVGAI Dataset.