

Joint Level Generation and Translation Using Gameplay Videos

by

Seyyede Negar Mirgati

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Seyyede Negar Mirgati, 2024

Abstract

Procedural Content Generation via Machine Learning (PCGML) faces a significant hurdle that sets it apart from other ML problems, such as image or text generation, which is limited annotated data. For example, many existing methods for level generation via machine learning specifically require a secondary representation beyond level images. However, the current methods for obtaining such representations are laborious and time-consuming, which contributes to the limited data problem. In this work, we aim to address the limited game level data problem by utilizing gameplay videos of human-annotated games to train a novel multi-tail framework to perform simultaneous level translation and generation. The translation tail of our framework can convert gameplay video frames to an equivalent secondary representation, while its generation tail can produce novel level segments. Evaluation results and comparisons between our framework and baselines suggest that combining the level generation and translation tasks can lead to improved performance for both tasks. Additionally, we have conducted experiments to evaluate the generalizability of our model across different scenarios. Our findings represent a possible solution to limited annotated level data, and we demonstrate the potential for future iterations of our model to generalize to unseen games.

Preface

This thesis has been accepted and published as a full paper in the IEEE Conference on Games 2023, as: Mirgati, Negar, and Matthew Guzdial. “Joint Level Generation and Translation Using Gameplay Videos.” 2023 IEEE Conference on Games (CoG). IEEE, 2023. Also, an extended version of it has been submitted to the IEEE Transaction on Games Journal.

Taking a new step, uttering a new word, is what people fear most.

– Fyodor Dostoevsky.

Acknowledgements

Firstly, I would like to express my sincere gratitude to Dr. Matthew Guzdial for providing me with invaluable guidance, support, and advice throughout my entire research process. I am truly fortunate to have had the privilege of conducting my graduate research under his mentorship, and I owe much of my academic growth and success to his expertise and dedication.

I would also like to thank Dr. Behnam Bahrak for his inspiring discussions and guidance throughout my B.Sc. His enduring influence continues to be a guiding inspiration in both my academic pursuits and personal goals.

Additionally, I am grateful for the support provided by the Canada CIFAR AI Chairs Program, which funded this work, and the Alberta Machine Intelligence Institute (Amii), whose support was instrumental in its completion.

Lastly, I would like to thank my family and friends for their love and encouragement, without which this achievement would not have been possible.

Contents

1	Introduction	1
1.1	Research Questions	3
2	Background	5
2.1	Deep Learning	5
2.1.1	Convolutional Neural Networks	6
2.1.2	GAN	6
2.1.3	VAE	7
2.1.4	VAE-GAN	8
2.1.5	VQ-VAE	8
2.2	Optical Flow	10
2.3	Level Translation	10
2.4	Level Generation	12
2.5	Effects of Training Data on Level Generation Methods	13
3	Joint Level Generation and Translation Using Gameplay Videos	14
3.1	Dataset	14
3.2	Model Architecture	15
3.3	Model Training	17
3.4	Hyperparameter Tuning	18
4	Experimental Setup and Results	20
4.1	Evaluation	20
4.1.1	Baselines	21
4.1.2	Metrics	22
4.2	Results	23
4.3	Experiments on Generalizability	27
4.3.1	Impact of Data Variation, Representation, and Embedding	28
4.3.2	Latent Space Analysis	29
4.3.3	Case Studies on Super Mario Land	32
4.3.4	Case Studies on Translation Accuracy	33
4.3.5	Case Studies on Translation of Natural Images	33
5	Conclusion	35
5.0.1	Limitations and Future Work	35
5.0.2	Conclusions	36
	References	37

List of Tables

4.1	Results of generation and translation evaluation metrics for our model and all baselines.	25
4.2	Results of translation evaluation metrics for variations of model architecture and representation techniques.	29
4.3	Evaluation results for Super Mario Land.	31

List of Figures

1.1	A frame from the game Kid Icarus with a hazardous tile highlighted by red.	2
2.1	Example of a simple Deep Neural Network.	6
2.2	The GAN architecture.	7
2.3	The VAE architecture.	8
2.4	VAE-GAN Framework.	9
2.5	VQ-VAE framework.	9
2.6	An example of sparse optical flow calculated for a frame of The Legend of Zelda. The red tracks show the recent movement path of the character.	11
3.1	Our model’s architecture.	16
4.1	Examples of the VAE baseline’s generated outputs.	25
4.2	KDE plots of the training dataset, demonstrating linearity vs. leniency for the top four models.	26
4.3	KDE plots of the test dataset, demonstrating linearity vs. leniency for the top four models.	26
4.4	Sample generations of our VAE-GAN. Samples 4.4a and 4.4b look more similar to Kid Icarus levels, whereas samples 4.4c and 4.4d are closer to the Super Mario Bros. levels. Samples 4.4e, 4.4f, 4.4g, 4.4h in the second row blend the characteristics of both Kid Icarus and Super Mario Bros. levels.	27
4.5	Sample translations of our VAE-GAN. Figures 4.5a (Super Mario Bros.) and 4.5b (Kid Icarus) belong to the test set of our dataset, and figures 4.5c (Duck Tales), and 4.5d (Megaman) are sample frames from unseen games.	28
4.7	Case studies of Super Mario Land data. The first row demonstrates two examples with high translation accuracy, while the second row shows two examples with the lowest translation accuracy.	31
4.8	Examples of our model’s translation of natural images.	33
4.9	Examples of our model’s most accurate (4.9a, 4.9c) and least accurate (4.9e, 4.9g) translations versus the real translations.	34

Chapter 1

Introduction

Procedural Content Generation (PCG) is the practice of using algorithms to generate new game content, as opposed to manual human creation. PCG techniques cover a broad range of content types, such as game levels, characters, maps, and quests. PCG can be used to assist game designers in the design phase and at runtime to generate content [40].

Historically, PCG was devised as a data compression method to address the problem of limited storage [45]. One of the first games that used PCG is the 1980 “Akabeth: World of Doom”, in which the user is asked to enter a number that is then used as a seed to procedurally generate the dungeons [1]. Since then, PCG techniques have greatly evolved and appear more commonly in modern games. One frequently cited modern game example is “No Man’s Sky,” in which almost all the game objects, such as creatures, planets, and solar systems, are procedurally generated [8].

PCG approaches can be categorized into three major types: constructive, search-based, and machine learning-based methods [44]. Constructive PCG [43] involves the use of hand-authored pieces of content and rules to put the pieces together to create new content. Search-based methods [55], on the other hand, explore a space of possible content and find desirable instances based on a fitness function. Finally, procedural content generation via machine learning (PCGML), which is the focus of this work, is the practice of generating new game content by applying various machine learning (ML) techniques to train on existing game data or environments and then sampling from trained models

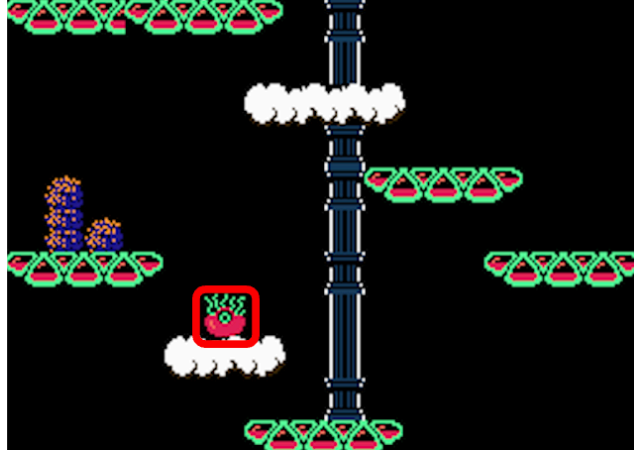



Figure 1.1: A frame from the game Kid Icarus with a hazardous tile highlighted by red.

to generate new content [14], [52].

Although there has been considerable advancement in the area, one of the fundamental challenges PCGML researchers face is the limited availability of annotated data. Contrary to other ML tasks, such as image generation, PCGML methods typically need more information than raw pixel representations. This additional requirement arises from the fact that game data, such as levels, have special structural features and functional rules, which are difficult or impossible to capture solely from images. For instance, consider the hazardous objects in the game Kid Icarus, such as the highlighted example in Figure 1.1. It is difficult to determine the behavioral properties of these tiles  (e.g., that they will kill the player) from their visual appearance alone. Consequently, we require additional information to generate new game content that closely follows the same structure and abides by the constraints of the original data [19].

One of the most commonly used approaches to represent this information for game levels is by translating tiles of pixels to text characters, where each character stands in for an object in the level. Each character is associated with a set of affordances, indicating which is harmful, which is breakable, and so on. The conventional procedure of generating string-based representations of game levels typically involves multiple iterations of image processing via tools

such as OpenCV [4], followed by manually checking the results, which is quite labor-intensive [13]. In this regard, the Video Game Level Corpus (VGLC) [53] has immensely assisted PCGML researchers by providing public access to string representations of 428 levels from 12 different 2D games. Though this is an impressive range of coverage in the area of games, it is still orders of magnitude smaller than numerous widely-used datasets (e.g., ImageNet [9], and CelebA [26]), used in image generation. If we could access sufficient data, we may be able to achieve the success and generalizability found in other modern machine learning domains.

A less explored yet promising secondary source of data for PCGML is gameplay video. The active community of video game enthusiasts has produced countless gameplay videos across different genres of games. This includes platformers that are currently the target of a large proportion of PCGML research. Despite the existence of significant video data, few approaches leverage gameplay video for game level generation. The existing methods either focus on a single game [13] or do not focus on level design [42].

In an attempt to tackle the problems mentioned above, we focus on a novel machine learning framework for simultaneous video-to-level generation and translation. In this context, “level generation” involves the creation of new game level segments for 2D platformer games. On the other hand, “level translation” refers to the process of converting existing game level content from an RGB pixel representation to a string-based representation that can be used for level generation. Our framework seeks to leverage commonalities in the learned latent representation between these processes by simultaneously addressing both level generation and translation tasks.

1.1 Research Questions

Motivated by the problems discussed earlier in this chapter, we propose the following research questions:

- RQ1) Can learning level generation and translation tasks in tandem benefit any of these tasks, compared to learning each task separately?

- RQ2) To what extent can this idea improve or hinder the performance quality of the proposed architecture regarding both generation quality and translation accuracy?
- RQ3) What is a machine learning-based architecture that can be employed to perform these tasks in a joint manner?
- RQ4) How can such an approach potentially address the problem of insufficient annotated data for procedural level generation via machine learning?

To answer the above research questions, we propose a novel machine learning model for simultaneous video-to-level translation and generation. We believe this can lead to an improved generation and/or translation performance, as these tasks are related. Our final model translates gameplay video into a tile-based string representation and simultaneously generates new level segments. We accomplish this with a modified VAE-GAN architecture, with each tail handling one task. We train our approach with YouTube videos and the VGLC level corpus but demonstrate the ability to generalize over content outside of the VGLC representation.

The rest of this thesis is organized as follows: In chapter 2, we overview necessary background information and briefly outline related work to understand this work. Chapter 3 explains our joint level generation and translation approach in detail. In chapter 4, we describe our evaluation methods and present the results. Finally, chapter 5 concludes this thesis by discussing the limitations of our approach and potential avenues for future work.

Chapter 2

Background

This chapter introduces the required technical knowledge for the following chapters related to the approaches we employ to construct our model and baselines. We provide background on Deep Learning and Deep Neural Networks (DNNs) such as Variational Autoencoders (VAEs), Vector-Quantized Variational Autoencoders (VQ-VAEs), and Generative Adversarial Networks (GANs), as we have made use of these techniques in the design of our model and baselines. We then discuss Optical Flow methods for tracking objects' motion in videos. To conclude this chapter, we cover related work on level generation and translation, the main tasks we focus on in this thesis. Subsequently, we explore existing research on the implications of varying amounts of training data on level generation methods.

2.1 Deep Learning

Deep learning is a subset of machine learning algorithms that are inspired by the human brain [35]. Deep learning relies on deep neural networks (DNNs) that consist of artificial neurons. Each artificial neuron has weights, biases, inputs, and outputs and can represent a mathematical function. An example of a simple DNN is represented in Figure 2.1. This DNN consists of an input layer with two neurons, two hidden layers, each with four neurons, and an output layer with two neurons.

One of the most essential advantages of deep learning approaches compared to other machine learning methods is the reduced need for human interven-

tion. Deep learning methods can work with unstructured data such as text and images. In this research, we make use of multiple widely-used deep learning architectures, namely generative adversarial networks, variational auto-encoders, and convolutional neural networks, to learn our desired tasks from image and string-based data.

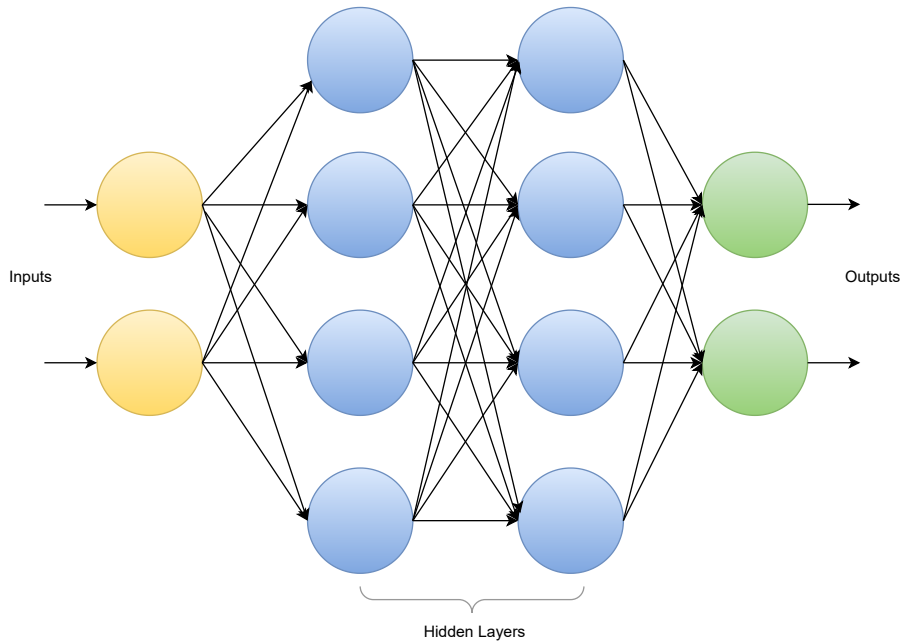


Figure 2.1: Example of a simple Deep Neural Network.

2.1.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of deep learning network that is mainly applied to two-dimensional data types such as images [24]. CNNs typically consist of a series of convolutional layers, activation functions, and pooling layers. The automatic learning of filter weights enables CNNs to extract useful features from images with a reduced need for human intervention. CNNs are widely used to perform various tasks such as image classification, object detection, and image segmentation.

2.1.2 GAN

A Generative Adversarial Network (GAN)[11], as shown in Figure 2.2, is a DNN that consists of a generator and a discriminator that compete against

each other. The generator’s goal is to output content that is hard for the discriminator to distinguish from the original content. On the other hand, the discriminator aims to strengthen its ability to differentiate the generator’s outputs from the real data. The generator and discriminator are trained in an alternating manner until convergence.

In this research, we make use of the GAN architecture as part of our model due to the effect it has on the latent space. Additionally, we leverage the Wasserstein GAN (WGAN) [2], which is a variant of the vanilla GAN that uses the Earth-Mover’s distance (EM) instead of the Jensen-Shannon divergence. The Earth-Mover’s distance is a metric that measures the minimum amount of work needed to move a distribution’s mass to another distribution [34], while the Jensen-Shannon divergence is a simpler method for calculating the similarity between two distributions.

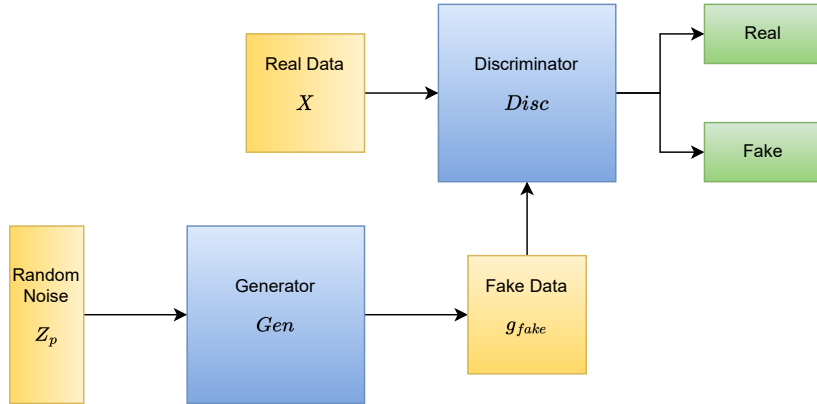


Figure 2.2: The GAN architecture.

2.1.3 VAE

Variational Autoencoder (VAE) [22], as shown in Figure 2.3, is a probabilistic generative model that consists of an encoder and a decoder. The encoder receives X as input and outputs a latent distribution represented by a mean μ and variance σ . The decoder then samples from this distribution to output \tilde{X} , a reconstruction of X . The model is trained with the objective of minimizing reconstruction loss as well as maintaining a Gaussian structure in the latent space. In this thesis, we make use of the VAE architecture as part of our model

architecture since our goal is to propose a model that can generate new data and perform the translation task.

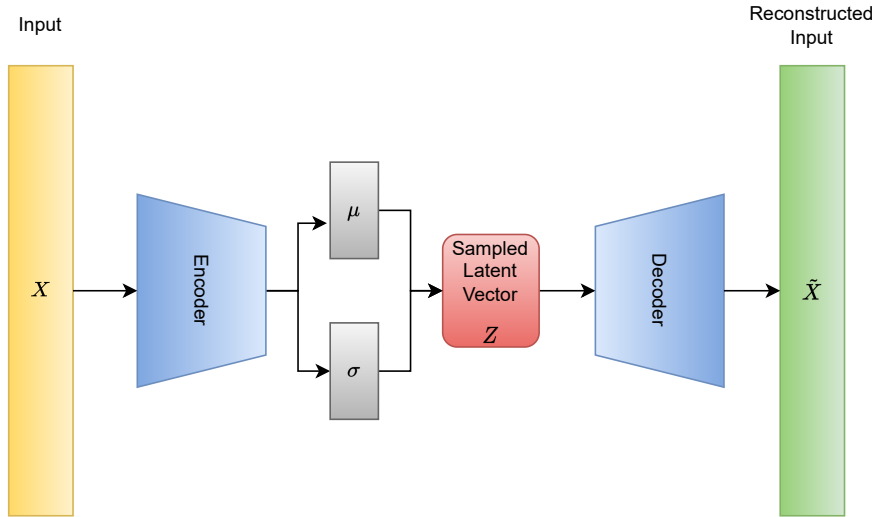


Figure 2.3: The VAE architecture.

2.1.4 VAE-GAN

A VAE-GAN [23], as shown in Figure 2.4, is a deep learning architecture that combines a VAE and a GAN by unifying the VAE’s decoder with the GAN’s generator. This means that a VAE-GAN is capable of encoding, generating, and differentiating between data samples. Larsen et al. [23] also provided evidence that the VAE-GAN architecture outperforms the individual VAE and the GAN architectures regarding the quality of generated images, providing motivation to apply it to our problem. To the best of our knowledge, this architecture has not been previously applied to a level translation task outside of the work presented in this thesis.

2.1.5 VQ-VAE

Vector-Quantized Variational Autoencoders (VQ-VAEs) are a subset of VAEs that employ Vector Quantization (VQ) to learn a discrete latent space. VQ is a dictionary-learning method that attempts to map the embedding vectors to symbolic encoded values. The VQ-VAE architecture has three main compo-

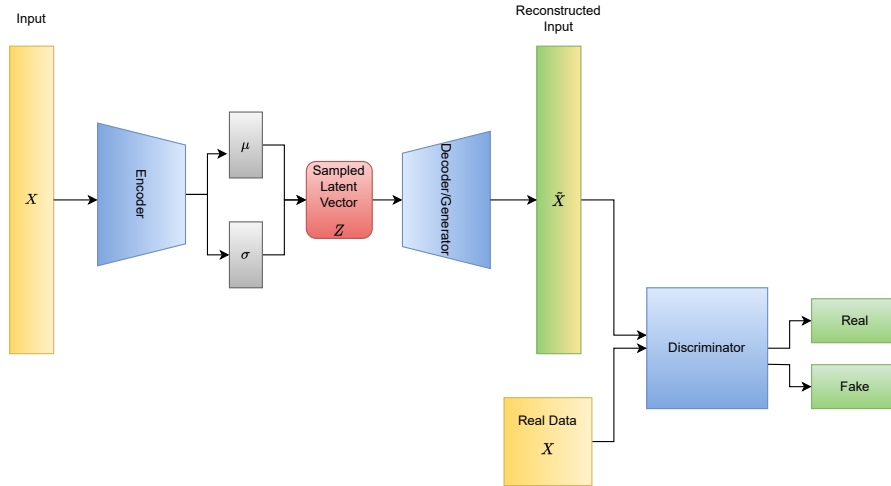


Figure 2.4: VAE-GAN Framework.

nents, as shown in figure 2.5: an encoder, a learnable codebook, and a decoder. The input data is first fed through the encoder to output $z_e(x)$. Next, the codebook converts $z_e(x)$ to a table of indices. This table is then converted to $z_q(x)$ by translating indices to the codebook embeddings indices in the codebook. Finally, the decoder receives $z_q(x)$ to reconstruct the input data.

In this thesis, we train and evaluate a variation of our model with the VQ-VAE architecture as one of its main components, as it may be a natural fit for our use case with discrete text data.

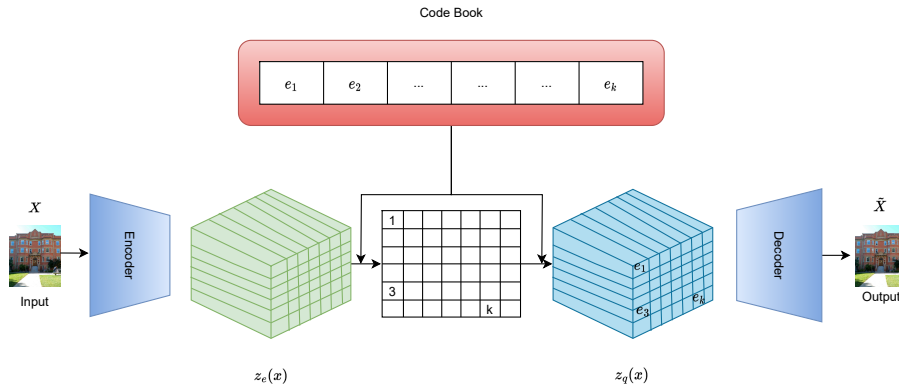


Figure 2.5: VQ-VAE framework.

2.2 Optical Flow

Optical Flow is an algorithm that detects the motion of objects, edges, and surfaces between two consecutive frames as a result of relative movement between the object and the camera. [16]. Optical flow has applications in different areas, such as traffic monitoring, image stabilization, face tracking, and robot navigation.

There are two main variants of optical flow: dense and sparse optical flow. In sparse optical flow, the motion is tracked for only the points of interest (such as object corners) in the frame. These points of interest can be automatically detected by algorithms such as Harris [15] and Shi-Tomasi corner detection [41]. Dense optical flow, on the other hand, tracks the movement of all the pixels in the frame. This means that the dense variant is more accurate but less computationally efficient than sparse optical flow.

In this research, we make use of the Lucas-Kanade implementation of sparse optical flow [27] with Shi-Tomasi corner detection, which is a commonly used robust and efficient algorithm in the field [25]. We believe adding the player character’s movement path to the data representation might be helpful to our model since it can provide additional help in terms of distinguishing foreground from background elements.

Figure 2.6 demonstrates an example of applying the Lucas-Kanade optical flow algorithm to the frames of gameplay video for the game The Legend of Zelda. The red dots in this image demonstrate the detected points by the algorithm. Also, the red tracks show the player character’s movement in consecutive frames as detected by the algorithm. In this specific example, the algorithm has only detected the player’s movement as opposed to the other elements in the background, which is a desirable result for our use case.

2.3 Level Translation

Level translation is the process of converting the raw pixel representation (image) of a game level into a secondary representation that is useful for down-

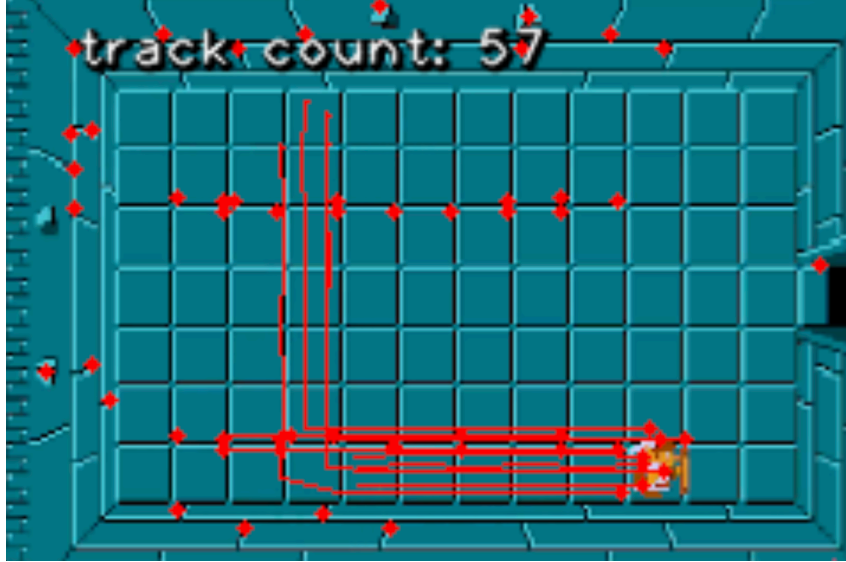


Figure 2.6: An example of sparse optical flow calculated for a frame of The Legend of Zelda. The red tracks show the recent movement path of the character.

stream tasks. In this thesis, we focus on level translation since it has the potential to address the issue of limited annotated data.

As an example, Summerville et al. [53] created the VGLC corpus that contains level images and their corresponding string representations. In the level images, each 16×16 pixel tile specifies a game object (e.g., enemy, door, coin), and the VGLC representation converts each tile to a character. As a result, it provides a sequence of characters (i.e., string) for each level image. It is most likely that the creators of the VGLC corpus utilized a non-ML approach involving the OpenCV library [4], and human editing passes to perform the tile-to-character translation. Although we use the VGLC representation in this work, we propose employing an ML model to translate from pixels to the character representation automatically.

Chen et al. [6] leveraged a Convolutional Neural Network (CNN) to translate 32×32 level segments to their corresponding string representation. They iteratively performed this process to convert an entire image to its equivalent string representation. Instead, we employ a Variational Autoencoder (VAE) to translate each video frame to its equivalent string representation at once.

Smirnov et al. [42] introduced a novel framework that can extract recur-

ring sprites from gameplay videos. Their translated representation was not intended for level design tasks, unlike our own, and instead focuses on game reconstruction and scene understanding.

Snodgrass and Ontañón [47] introduced an approach to transform levels from one domain to levels of another by translating the input domain’s tiles to the target domain. In comparison, we design a model that learns to translate level images to the equivalent string representation in the same domain. Having said that, we do train our approach on multiple domains, meaning that it is capable of generating similarly “blended” levels.

2.4 Level Generation

Researchers have suggested various methods for generating levels through the use of PCGML [52]. Most of these rely on access to a string-based representation (e.g., the VGLC), which makes them only applicable to a very limited number of games for which such representation is available [52]. We do not cover the majority of work that relies on these representation as we have different aims. Relevant to our work, Guzdial and Riedl [13] proposed an approach utilizing gameplay video to generate new levels. While their model successfully captured the style of the original game of training data, it does not generalize to unseen games.

A number of existing approaches focus on generating blended levels. For instance, Sarkar, Yang, and Cooper [39] trained a VAE on two games to generate new blended level segments. Sarkar et al. [38] then built upon the approach proposed in [39] and expanded the input domain to six games. While our approach also trains on multiple domains and generates blended levels, this is not the primary focus of our work.

Jadhav and Guzdial [19], and Khameneh and Guzdial [21] focused on learning new representations that could be used for downstream tasks. These methods can be viewed as alternative strategies for dealing with the problem of insufficient training data in PCGML. We differ from these two approaches in the use of video data and in our unique VAE-GAN-based architecture that learns

to translate video frames and generate new level segments simultaneously.

2.5 Effects of Training Data on Level Generation Methods

Most work on procedural level generation via machine learning make use of all available data without considering the implications of varying amounts of training data on model performance [13], [48], [51]. Snodgrass, Summerville, and Ontañón [49] studied the impact of using different numbers of Super Mario Bros. levels to train an LSTM-based and a Multi-dimensional Markov chain-based model. The evaluation results of this work showed that choosing a smaller subset of available levels may be beneficial for these models' performance. In this thesis, we explore the effect of different numbers of training games on the translation performance of our model.

Chapter 3

Joint Level Generation and Translation Using Gameplay Videos

In this thesis, we seek to train a single model on gameplay video capable of both level generation and translation. This chapter describes the four parts of the procedure for implementing and training such a model. The first part outlines the primary steps we undertook to collect our dataset. The second part provides a detailed explanation of our model architecture. The third part covers the training process of our model, and the last part discusses details of the hyperparameter tuning of our model.

3.1 Dataset

We required a dataset that matched gameplay video frames to level structure for our problem. To create this dataset, we undertook the following steps:

1. We obtained high-quality gameplay videos of Super Mario Bros. and Kid Icarus from YouTube.
2. We used a video parser script ¹ to parse each video into a series of frames. We used 2 frames per second (FPS) for Super Mario Bros. and 1 for Kid Icarus. This helped keep multiple frames with the exact same level structures out of our dataset.

¹<https://github.com/mguzdial3/VideoParser>

3. We resized all frames so that the tile size was 16×16 pixels, which is the same as the tile size of level images in the VGLC.
4. We used the VGLC’s level images to match the gameplay frames to the VGLC tile-based string representation. We employed OpenCV’s template matching algorithm to look for the closest match of each frame in its corresponding level image. Using the location of the closest match, we paired the frame data with the appropriate string representation.

As the VGLC string representations differ between Super Mario Bros. and Kid Icarus, following the prior work [19], [38], we unified the string representations by using a common set of 9 tile types: $\{(\#: \text{ solid, ground}), (-: \text{ empty, background}), (D: \text{ pipe, door}), (H: \text{ enemy, harmful}), (M: \text{ moving}), (T: \text{ solid-top}), (B: \text{ block}), (S: \text{ breakable}), (O: \text{ collectible})\}$.

5. We chose 10×15 tiles as the string representation size and cropped our frame images to match the corresponding translations. We chose this string size since this was the smallest common window size that would work with the VGLC games.
6. Finally, we downsized all frame images to 75×50 pixels since this frame size retained enough pixel information for the tiles to be recognizable and allows for odd-sized filters in the convolutional layers of our model. We split our final dataset into training and test sets by setting aside level 6 of Kid Icarus and worlds 7 and 8 of Super Mario Bros. as the test data. In addition, we upsample the Super Mario Bros. level segments in our training set to have an equal number of samples from both games. Our final dataset consists of 3956 (frame, string translation) pairs, half for each game, in the training set and 360 pairs in the test set.

3.2 Model Architecture

Our architecture is based on a typical VAE-GAN architecture introduced by [23]. As with all VAE-GAN architectures, it is made up of the VAE and GAN components. Our VAE component is primarily responsible for translating

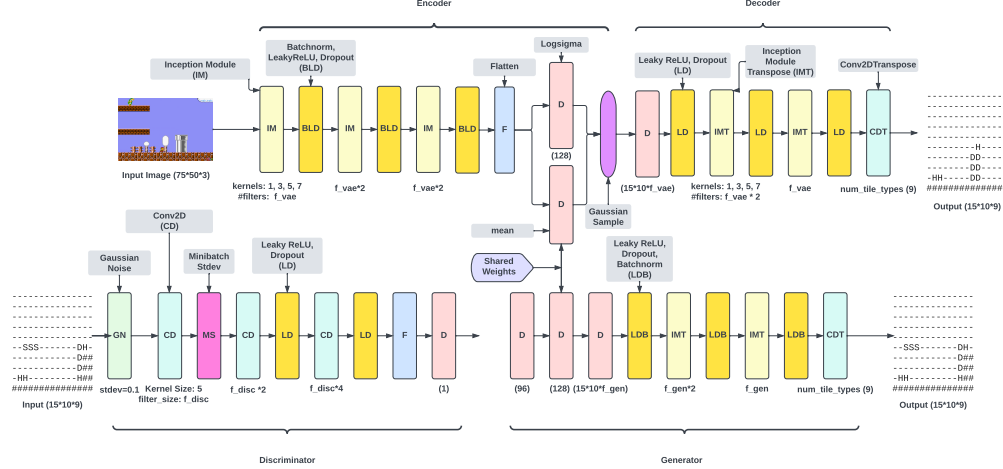


Figure 3.1: Our model’s architecture.

frames into a tile-based string representation. Meanwhile, the GAN component is in charge of generating new level segments. Figure 3.1 demonstrates our model architecture. The defining characteristic of our VAE-GAN is the split of the generator and the decoder and the shared weights between the encoder and generator, which we included to help the model learn a latent space that would benefit both tasks.

Our model’s VAE has two parts - an encoder and a decoder. The encoder receives frame images of size $75 \times 50 \times 3$ as input, where 3 stands for the number of RGB channels. Our encoder is made up of consecutive Inception modules [54], followed by batchnorm [18], leaky ReLU, and dropout layers. After all these layers, our encoder has a shared dense layer, and a variance dense layer, both of which have d neurons, where d represents the size of the latent space. Finally, the last layer of our encoder uses the mean μ and the variance σ from the previous layers to sample a d -dimensional vector from the latent space. Our decoder receives the d -dimensional vector as its input from the encoder. It then passes it through consecutive sets of transposed Inception modules, leaky ReLU, and dropout, followed by a Convolutional Transpose layer with nine filters. The output of our decoder, which has size $10 \times 15 \times 9$, is a reconstruction of the input in the tile-based string representation.

For our model, we make a modification to the original VAE-GAN architecture. Our GAN consists of a separate generator and a discriminator. To

elaborate further, instead of having a single unified decoder/generator, we use separate decoder and generator networks to avoid overcomplicating the job of the single decoder/generator network. However, we allow the VAE and GAN components to share information by sharing the weights of the encoder’s mean layer with the generator’s second dense layer. Apart from the shared layer and an initial additional dense layer to facilitate the weight sharing, our generator has a very similar architecture to the decoder and the same input and output sizes.

Finally, the discriminator network consists of a Gaussian noise layer, a convolutional layer, a minibatch standard deviation layer [20], and two consecutive convolutional layers followed by leaky ReLU and dropout layers, and a final single-neuron dense layer. Notably, the last dense layer doesn’t have any activation functions because our GAN follows the properties of the WGAN variation. The input size of this network is $10 \times 15 \times 9$, and it outputs a 1-dimensional value representing the model’s evaluation score for the input.

3.3 Model Training

Our VAE-GAN architecture has a unique optimization process. We train our model as a whole, meaning that our VAE and GAN components are trained in a joint manner. In each epoch, we go through three primary stages. Firstly, we train our encoder and decoder to learn to perform the frame translation task. Secondly, following the training procedure of WGAN with gradient penalty [12], we train our discriminator towards optimality by training it for ten steps. Finally, in the last stage, we train our generator for a single step.

We refer to algorithm 1 (adapted from [23]) for an overview of the training process.

Our model is implemented using the Keras library [7]. The training hyperparameters of our models are as follows:

- Alpha coefficient of all leaky ReLU layers: 0.1473
- Discriminator number of CNN filters(f.disc): 8

- Generator number of CNN filters (f_gen): 2
- VAE number of CNN filters (f_vae): 2
- Discriminator dropout rate: 0.4684
- Generator dropout rate: 0.2400
- VAE dropout rate: 0.2426
- Latent space size: 128
- GAN learning rate: $1e - 4$

We obtained these parameters by tuning the GAN and the VAE. The tuning process is covered in more detail in the subsequent section.

3.4 Hyperparameter Tuning

Since GANs are sensitive to hyperparameter changes and hard to tune, we developed an approach to find a shared set of hyperparameters for all our models. We tuned our VAE and GAN baselines using the same approach and employed the discovered hyperparameters for the VAE-GAN models. We utilized Keras Tuner’s Bayesian Optimization function with 10 trials and 50 epochs for each trial. Furthermore, we employed the following approach to estimate the optimization objective that our tuning process minimizes. First, we generated 100 outputs and then computed a mean squared error (MSE) for each output by comparing it against all the training samples and finding the closest match. We then averaged the MSE values of the generated samples to calculate the final score. Notably, our goal for designing this method was to enable the tuning process to avoid choosing hyperparameters that led to the generation of obviously poor outputs (e.g., all background) that we observed while developing our models.

Algorithm 1 Our VAE-GAN with gradient penalty. We use default values of $n_{disc} = 10$, $\lambda = 10$

Initialize network parameters $\leftarrow \theta_{Enc}, \theta_{Dec}, \theta_{Gen}, \theta_{Disc}$
repeat

// Stage 1: VAE Training

$(X, Y) \leftarrow$ random mini-batch from dataset

$Z \leftarrow Enc(X)$

$\mathcal{L}_{prior} \leftarrow D_{KL}(q(Z|X)||p(Z))$

$\tilde{X} \leftarrow Dec(Z)$

$\mathcal{L}_{reconstruction} \leftarrow Categorical_Cross_Entropy(\tilde{X}, Y)$

$\theta_{Enc} \stackrel{+}{\leftarrow} -\nabla_{\theta_{Enc}}(\mathcal{L}_{prior} + \mathcal{L}_{reconstruction})$

$\theta_{Dec} \stackrel{+}{\leftarrow} -\nabla_{\theta_{Dec}}(\mathcal{L}_{prior} + \mathcal{L}_{reconstruction})$

// Stage 2: Discriminator Training

for $t = 1, \dots, n_{disc}$ **do**

$Z_p \leftarrow$ samples from prior $N(0, I)$

$g_{fake} \leftarrow Gen(Z_p)$

$d_{fake} \leftarrow Disc(g_{fake})$

$d_{real} \leftarrow Disc(Y)$

$gp \leftarrow gradient_penalty(Y, g_{fake})$

$\mathcal{L}_{Disc} \leftarrow mean(d_{fake}) - mean(d_{real}) + \lambda * gp$

$\theta_{Disc} \stackrel{+}{\leftarrow} -\nabla_{\theta_{Disc}} \mathcal{L}_{Disc}$

end for

// Stage 3: Generator Training

$Z_p \leftarrow$ samples from prior $N(0, I)$

$g_{fake} \leftarrow Gen(Z_p)$

$d_{fake} \leftarrow Disc(g_{fake})$

$\mathcal{L}_{Gen} \leftarrow -mean(d_{fake})$

$\theta_{Gen} \stackrel{+}{\leftarrow} -\nabla_{\theta_{Gen}} \mathcal{L}_{Gen}$

until deadline

Chapter 4

Experimental Setup and Results

Generative models can learn the underlying structure of input data and generate new data instances. Unlike previous approaches that accomplish level generation in a sequential manner, we present a method that is able to perform joint level generation and translation using gameplay video frames. In this chapter, we use our modified VAE-GAN-based model to learn to perform simultaneous level generation and translation. Using various evaluation methods, we show that our approach can achieve a better performance regarding translation accuracy and generation quality. Additionally, we conduct further experiments to shed more light on the generalization ability of our model.

4.1 Evaluation

The purpose of our model is to accomplish simultaneous level generation and translation. The ideal way to evaluate the generative quality of a level generator is to conduct a human study, where designers/players inspect the generated outputs and assess them. However, this would be premature for an initial evaluation of this approach. Thankfully, researchers in the field of PCGML have proposed several metrics to address situations like this that do not require human evaluation [3], [5], [17], [30], [46], [50]. On the other hand, we do not require human evaluation for the translation task as we can utilize metrics like accuracy and f1-score to assess the model’s performance. The following subsections present the baselines and metrics we chose for assessing our model.

4.1.1 Baselines

To evaluate the performance of our proposed model, we used the following existing architectures for the level generation task:

- **VAE-GAN:** We made use of the original VAE-GAN architecture [23]. Therefore, the primary difference between this baseline and our model is that this baseline only has one decoder/generator component instead of separate modules. Moreover, to make the comparison between architectures clearer, we leveraged the same reconstruction loss as our VAE-GAN. Noteworthy, we used $1e - 6$ as the reconstruction vs. generation coefficient since this value was used in the original VAE-GAN paper [23].
- **GAN:** For this baseline, we removed the VAE components from our model and kept the generator and discriminator. The main difference between this GAN and our model’s GAN module is that it no longer has a shared layer.
- **VAE:** As above, but removing the GAN components of our model.
- **VAE-GAN_TEXT, VAE_TEXT:** Same as the VAE-GAN and the VAE, but with the VGLC string representation of frames as input instead of frame images.

All of our baselines represent variations of our architecture, which will allow us to determine the utility of our VAE-GAN to this task in comparison to a typical VAE-GAN and the individual GAN and VAE components. In addition, we include variations of these baselines trained only on VGLC data to determine the utility of employing gameplay video data. We trained all our baselines using a batch size of 8 for 300 epochs, as empirical evidence showed that 300 epochs were enough for our models to converge. Moreover, we kept our baselines as similar as possible by utilizing the same hyperparameters as our model.

4.1.2 Metrics

Following previous research [19], [37], we adapted the following metrics proposed by prior PCGML work to evaluate our model’s generation and translation quality:

- **Linearity:** This metric measures how well a level conforms to a straight line. We employed the implementation of this metric from Jadhav and Guzdial [19].
- **Leniency:** This metric measures how hard a level is. We calculated this by subtracting the number of harmful/enemy tiles and half the number of moving tiles from the total number of tiles in the frame [37].
- **Interestingness:** This metric measures the number of interesting tiles in a level. To calculate this, we counted the number of doors, moving platforms, collectibles, and harmful tiles [37].
- **Playability:** The purpose of this metric is to determine whether a level segment is playable or not by trying to find a path from the lowest piece of level structure to the highest piece of level structure. We adapted this implementation of playability and pathfinding agents from Sarkar and Cooper [36], [37].
- **Accuracy:** We used this metric to evaluate our model’s performance regarding the ability to translate level segments. We calculated accuracy by dividing the number of correctly translated tiles by the total number of tiles in the level segment.

Additionally, we used the following metrics as means of comparing our model and baselines based on the values that we obtained using the metrics mentioned above:

- **Energy Distance (E-Distance):** The purpose of this function is to measure the distance between two distributions [32]. In this thesis, we considered the linearity, leniency, and interestingness of generated samples as features of their distribution to estimate how similar the outputs

of each model are to our dataset. We used the GeomLoss library [10] to calculate this metric.

- **Kernel Density Estimation (KDE):** This statistical approach is used to estimate the probability density function of a distribution based on a set of samples [33]. We utilized KDE plots to visualize the generative space of our models and compare them against both our training and test sets.

We note that to calculate the training metrics, we generated 3956 random samples for each model to have an equal number of samples as our training dataset. Similarly, we generated 360 random samples to compare our model’s outputs against the test dataset.

4.2 Results

Table 4.1 presents the results of the evaluation metrics for our model and all baselines. We bold the best value for a particular column or metric, and a dash indicates that the corresponding model does not have a defined value for that metric. To be more specific, best means highest values for the accuracy and playability columns, and lowest values for the e-distance columns.

Notably, our table has two playability columns, as we ran Super Mario Bros. and Kid Icarus pathfinding agents for all generated outputs. We made this decision because our model and baselines learn a single latent space, so they produce outputs covering both games and the space between them. The playability values in our table represent the portion of level segments that were playable according to the respective playability agent.

We observe that our VAE-GAN outperforms all the baselines regarding e-distance across both training and test datasets. This suggests that the generated outputs from our VAE-GAN are closer to the original datasets than the other baselines. Also, the test e-distance results provide evidence for our model’s superior ability to generalize. The test e-distance also demonstrates evidence of overfitting, as the text-based models perform poorly on it. We take

this as evidence that using gameplay video is beneficial for the level generation task. A possible explanation for this is that drawing on video data could have a similar effect as data augmentation in terms of introducing small variations on the same structures. For example, each second of Super Mario Bros. gameplay video produces two frames in our dataset (with $FPS = 2$). This results in some frame images with significant overlap, but with differences in their pixel representation due to the dynamic nature of gameplay video (e.g., movement of the player and enemies). In comparison, the static string representation had no such variation and no way to generate it automatically.

Considering the above analysis, it's important to acknowledge that we refrain from making a definitive claim regarding the superiority of our model over the original VAE_GAN, which achieves the closest results in terms of generation abilities. Given the sensitivity of GAN training to hyperparameters and random seeds [28], one could possibly discover a version that outperforms our model. Notably, we haven't explored the space of possible seeds or reconstruction vs. generation coefficients in this research.

In terms of reconstruction accuracy, our model achieves the best performance, tied with the VAE_TEXT model, though it is clearly superior in terms of generation quality. Furthermore, we performed statistical testing [58] using Python's SciPy library [57] on the accuracy results of our model against the baselines. The obtained p-values show that the reconstruction accuracy of our model is significantly greater than all the other baselines, except for the the VAE_TEXT.

Turning our attention to the playability results, we see that all the playability (SMB) percentages are quite close except for the VAE baseline. By inspecting this baseline's outputs, such as examples provided in Figure 4.1, we recognized that its outputs had less variety and simpler structures, leading to improved playability but an inferior e-distance. Furthermore, the playability (KI) results are similar and rather low. We anticipate that this is due to this Kid Icarus pathfinding agent [36], [37] only achieving 77% on the real Kid Icarus level segments of our dataset. The original VAE-GAN does outperform our approach for playability (KI) but not in terms of playability (SMB)

or other metrics. We take this as evidence of the utility of the VAE-GAN architecture generally and of our alterations to this architecture specifically.

Model	Training Accuracy	Test Accuracy	Training E-distance	Test E-distance	Playability (SMB)	Playability (KI)
Our VAE-GAN	0.94	0.88	0.31	0.39	0.86	0.54
Original VAE-GAN	0.90	0.85	0.37	0.44	0.85	0.60
GAN	-	-	0.64	0.83	0.85	0.54
VAE	0.94	0.87	1.69	0.84	0.94	0.58
VAE-GAN.TEXT	0.92	0.87	0.66	1.81	0.85	0.54
VAE.TEXT	0.92	0.88	2.53	2.55	0.85	0.47

Table 4.1: Results of generation and translation evaluation metrics for our model and all baselines.

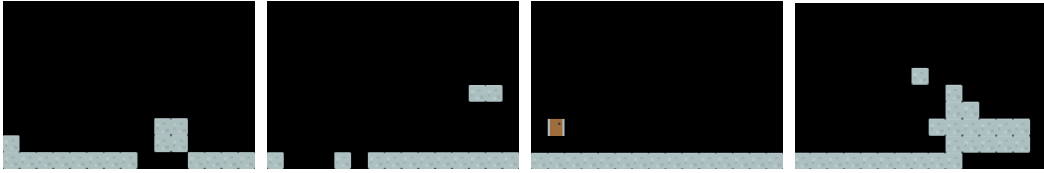


Figure 4.1: Examples of the VAE baseline’s generated outputs.

Figure 4.2 demonstrates linearity vs. leniency KDE plots for the top four models in terms of training e-distance. The first row of Figure 4.2 plots the linearity vs. leniency against all the Kid Icarus samples from our training dataset for our VAE-GAN, the original VAE-GAN, GAN, and the VAE-GAN_TEXT models, respectively. The second row plots the same metrics for the same models against our dataset’s Super Mario Bros. samples. We chose these architectures as they had the best results in Table 4.1. As we move from left to right in each row, we see a decreasing trend regarding coverage of the original distribution. We also observe that this decreasing trend of coverage correlates with the trend that we see in the e-distance values of these four models.

Figure 4.3 demonstrates linearity vs. leniency KDE plots for the top four models in terms of test e-distance. Similar to the training KDE plots, we can see the correlation between the e-distance trend and coverage of the test dataset in the KDE plots.

Figure 4.4 displays four instances of our VAE-GAN’s generated outputs using a common tile representation (obtained from Kenney ¹). Samples 4.4a and 4.4b have a similar structure as the Kid Icarus levels, while samples 4.4c

¹kenney.nl/assets/platformer-art-deluxe

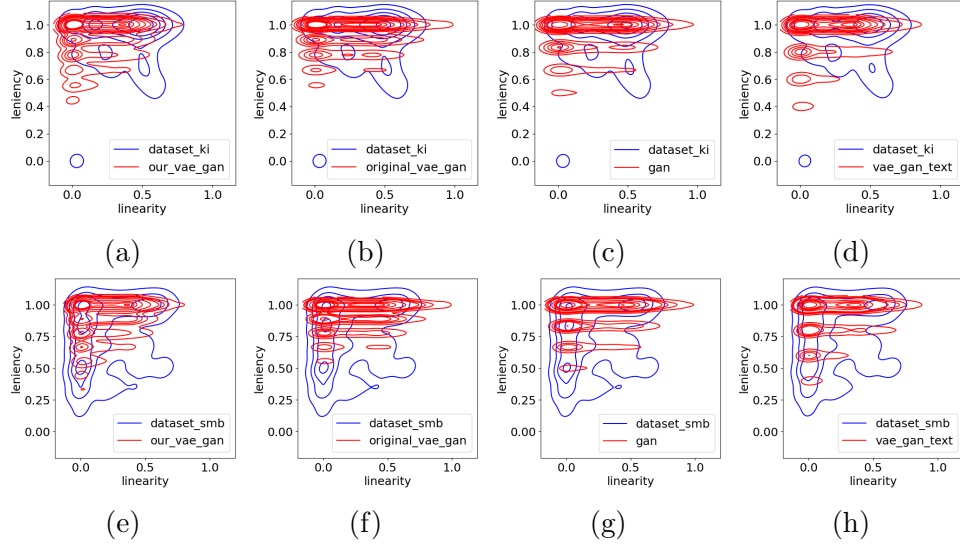


Figure 4.2: KDE plots of the training dataset, demonstrating linearity vs. leniency for the top four models.

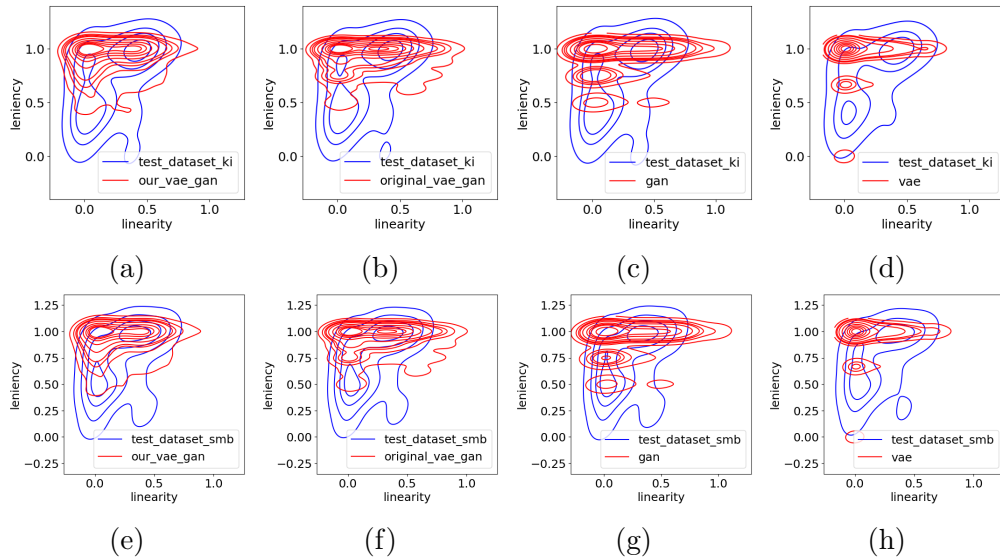


Figure 4.3: KDE plots of the test dataset, demonstrating linearity vs. leniency for the top four models.

and 4.4d have a closer resemblance to the Super Mario Bros. levels. Moreover, the sample images in the second row blend the characteristics of both games. We should note that we looked at a hundred randomly generated outputs to find the examples that were closest to these criteria. To elaborate on the blended aspects of examples in the second row, although sample 4.4e has solid-top platforms (cloud-like tiles) that are placed in a manner that is similar to

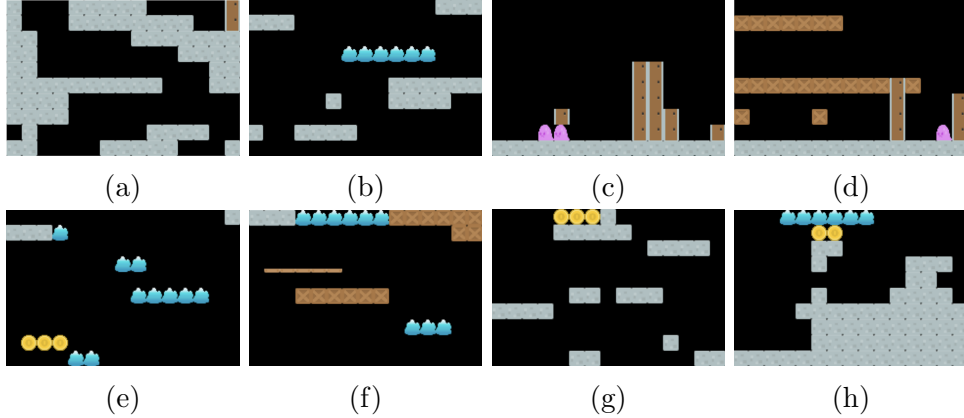


Figure 4.4: Sample generations of our VAE-GAN. Samples 4.4a and 4.4b look more similar to Kid Icarus levels, whereas samples 4.4c and 4.4d are closer to the Super Mario Bros. levels. Samples 4.4e, 4.4f, 4.4g, 4.4h in the second row blend the characteristics of both Kid Icarus and Super Mario Bros. levels.

Kid Icarus level structure, it has collectibles (coin-shaped tiles) that only occur in Super Mario Bros. levels. This observation indicates that our model can blend the aspects of both games to generate novel level segments that were not included in the training dataset.

Figure 4.5 presents sample translations of our model. The upper row consists of frames from Super Mario Bros., Kid Icarus, Duck Tales, and Megaman, respectively. The translation of each frame is below it. We observe that the translation performance is significantly better for the first two samples (4.5a, 4.5b) since our model saw samples from these two games during training. We selected example outputs to demonstrate this, but note from Table 4.1 that we had high translation accuracy.

4.3 Experiments on Generalizability

In this section, we cover experiments that we designed to better understand our models’ generalization ability. Subsection 4.3.1 explains the impact of data variation, representation, and embedding technique on the translation accuracy of unseen data. Subsection 4.3.2 provides an in-depth analysis of the latent space of our model, and subsections 4.3.3, 4.3.4, and 4.3.5 demonstrate case studies on translation performance of our model with Super Mario Land,

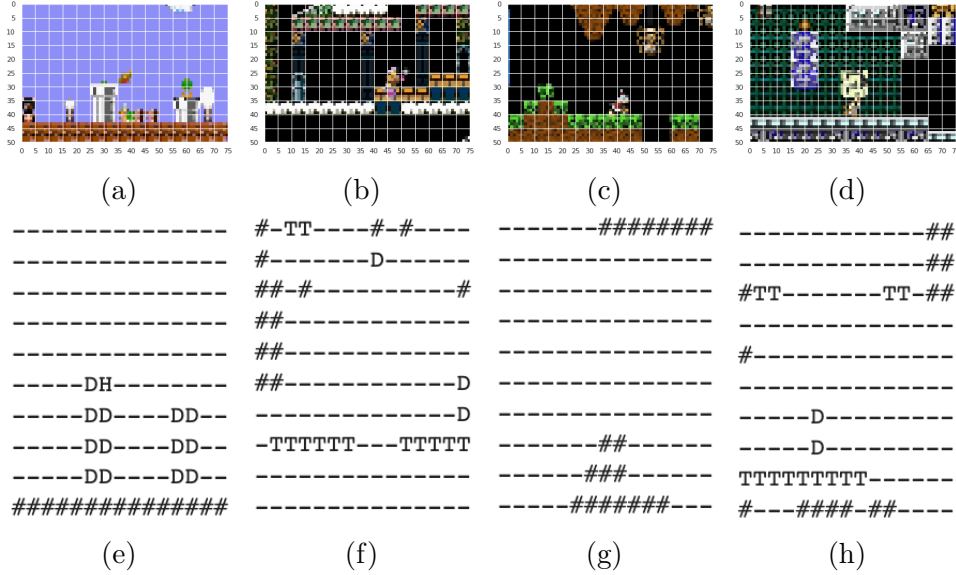


Figure 4.5: Sample translations of our VAE-GAN. Figures 4.5a (Super Mario Bros.) and 4.5b (Kid Icarus) belong to the test set of our dataset, and figures 4.5c (Duck Tales), and 4.5d (Megaman) are sample frames from unseen games.

Megaman, and natural images.

4.3.1 Impact of Data Variation, Representation, and Embedding

Table 4.2 demonstrates our evaluation results given variations of number of training games, latent space embedding, and data representation techniques. In this table, VAE-GAN refers to our VAE-GAN without any changes to the model architecture, while VQ-VAE-GAN refers to a variation of our model that uses a discrete latent space. Notably, in this evaluation, we have used Megaman’s data as an unseen game test dataset.

Comparison between the results of the first two rows of table 4.2 suggests that adding one additional game results in a drop in the translation performance of our model. We believe this is likely due to the substantial structural difference between the third training game (The Legend of Zelda), a dungeon crawler, and the test game, a platformer. The third training game further pushes the latent space away from that of the target data, leading to a performance drop.

Comparing the second and third rows of the table provides evidence that

adding frames’ detected edges as an additional channel to the input data can result in improved translation accuracy for unseen data. We believe this additional representation improves our model’s ability to generalize, as it is not dependent on the variations of color.

The fourth row uses the edge-detection representation technique with a VQ-VAE variation of our model, which doesn’t improve the results comparing the default setting (first row) or the best setting (third row). We attribute this to the discrete latent space of this model, which is less capable of capturing the variation in the input image data, resulting in reduced generalizability.

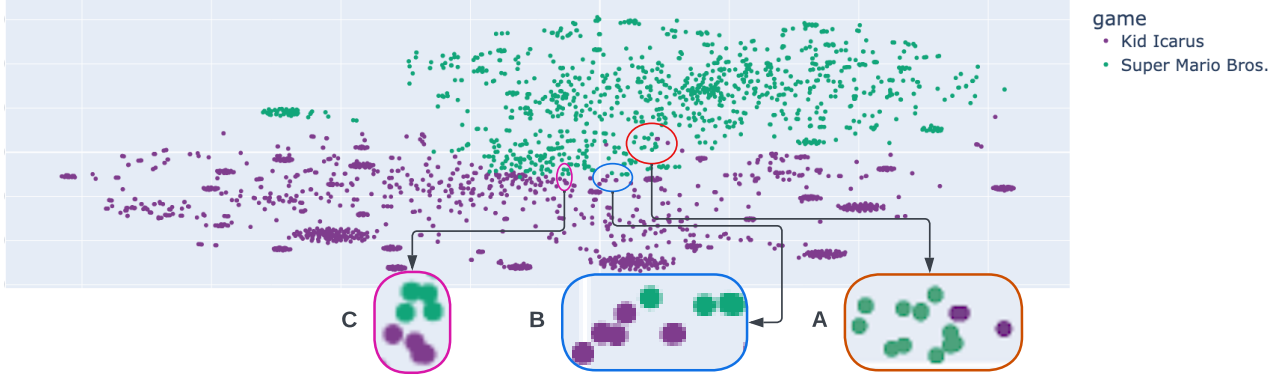
Lastly, the bottom row leverages both edge detection and optical flow information as the data representation techniques. Notably, this experiment results in a 1% accuracy improvement for training data compared to the third row. We believe this is probably due to the fact that the background is simpler and has fewer decorative elements in Mario and Kid Icarus. Hence, employing optical flow on their dataset yields a more discernible separation between the player’s motion and extraneous background elements, which ideally should not be included in the flow tracking.

Model	Training Games	Data Representation	Training Accuracy	Test Accuracy	Test Macro F1-score
VAE-GAN	KI, SMB	-	0.94	0.59	0.12
VAE-GAN	KI, SMB, LoZ	-	0.91	0.55	0.11
VAE-GAN	KI, SMB	Edge Detection	0.93	0.63	0.15
VQ-VAE-GAN	KI, SMB	Edge Detection	0.93	0.55	0.13
VAE-GAN	KI, SMB	Edge Detection, Optical Flow	0.94	0.60	0.13

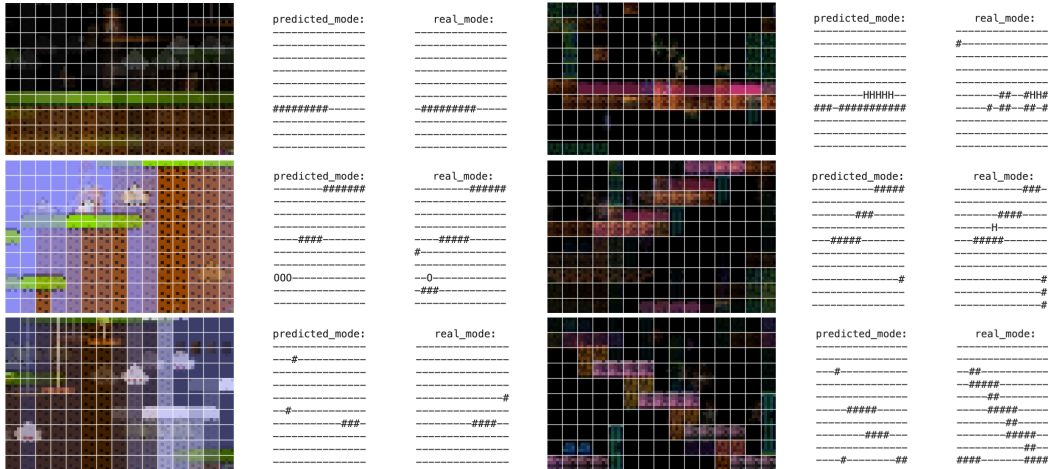
Table 4.2: Results of translation evaluation metrics for variations of model architecture and representation techniques.

4.3.2 Latent Space Analysis

Our model has a unique architecture that shares the embedding layer weights between its VAE and GAN tails. Motivated by this design choice, we conduct a closer inspection of the embedding space using t-SNE to determine whether any semantic groups emerged. T-Distributed Stochastic Neighbor Embedding (t-SNE) is a commonly-used statistical method used to visualize high-dimensional data in a two or three-dimensional space [29]. We used the



(a) Examples of types of overlap between the two game clusters. Region A indicates the case where datapoints are closer in terms of visual characteristics rather than translations, while region B demonstrates the opposite case. Region C showcases an example where datapoints are dissimilar in both comparison metrics.



(b) Average images and mode of translations for datapoints belonging to regions A, B, and C. The first row demonstrates pixel-wise average images of SMB datapoints versus the Kid Icarus datapoints and the character-wise mode of their predicted and real translations for region A. The second and third rows display the same information calculated for regions B and C, respectively.

Scikit-learn implementation of the t-SNE algorithm [31], with random initialization, perplexity of 100, and an automatic learning rate.

Figure 4.6a demonstrates the t-SNE visualization of our VAE-GAN’s latent space. There are two noticeable large clusters, one containing Kid Icarus datapoints and the other containing Super Mario Bros. datapoints. While these clusters exhibit separation overall, we observe some intersections. Therefore, we analyzed the overlapping areas and detected three different patterns. Notably, for each overlapping region marked in Figure 4.6a, we partitioned the

points into two categories: Super Mario Bros. and Kid Icarus. For each category, we computed the pixel-wise average of RGB values across the images of datapoints and determined the character-wise statistical mode of their corresponding string-based translations.

The first pattern (A) represents cases where the two groups of datapoints have more similarities in terms of visual appearance compared to the string-based translations. The second pattern (B) exhibits cases where datapoints are more similar in terms of string-based translation compared to the visual features in their images. Finally, the last pattern (C) identifies a region where the two groups look very different regarding the visual appearance of average images and average string-based representations. These examples demonstrate that our model makes use of both visual features and the tile-based structure to capture the underlying data distribution.

	Precision	Recall	F1_score	Support
#	0.59	0.32	0.42	1486
-	0.82	0.93	0.87	5930
D	0.32	0.04	0.07	151
H	0.00	0.00	0.00	36
M	0.00	0.00	0.00	0
T	0.00	0.00	0.00	0
B	0.00	0.00	0.00	35
S	0.00	0.00	0.00	255
O	0.07	0.02	0.03	57
accuracy			0.75	7950
macro avg	0.20	0.15	0.15	7950
weighted avg	0.73	0.75	0.73	7950

Table 4.3: Evaluation results for Super Mario Land.

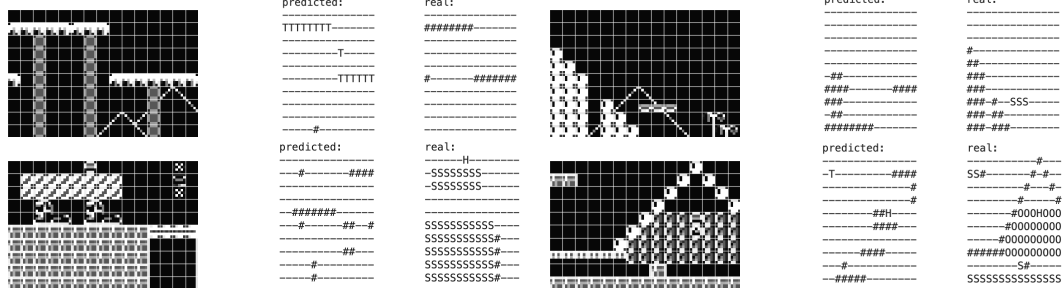


Figure 4.7: Case studies of Super Mario Land data. The first row demonstrates two examples with high translation accuracy, while the second row shows two examples with the lowest translation accuracy.

4.3.3 Case Studies on Super Mario Land

In this work, we have designed an automatic level generation and translation method that learns to generate new level segments based on visual input. Inspired by previous work [6], we decided to determine whether our approach can be adapted to perform co-creative level generation. One possible strategy to achieve co-creativity for our model is to feed human-drawn sketches to it and obtain reconstructed tile-based level segments that conform to the patterns of the input sketch. Since we did not have access to any sketch-to-tile datasets, we used Nintendo’s Super Mario Land levels from the VGLC dataset. Levels of this game are black-and-white and much less detailed compared to the Super Mario Bros. and Kid Icarus data we used to train our model. While this data may not represent an ideal choice, it is the best we could find in terms of resemblance to sketches and platformer games.

We processed the three World 1 levels from Super Mario Land to obtain 53 samples. Additionally, we inverted the colors of these input images as this made their visual appearance closer to the training data and resulted in better translations.

Table 4.3 represents the evaluation results of our original model’s translation of this dataset. Our model achieves a reasonable performance for the background (-) and solid (#) tile types, but struggles with correctly identifying the less frequent tile types. We attribute this to the fact that these tile types appear much less frequently in our training data, making it harder for our model to correctly identify them in unseen games.

Figure 4.7 demonstrates some of the best and worst translations of our model in terms of accuracy. We observe signs of overfitting as the structures are being mapped to the closest Mario or Kid Icarus-like structures.

Notably, we did not train our model to perform this specific task, so we do not expect perfect performance. However, the evaluation results provide evidence for the potential ability of our model to perform reasonably well at this task.

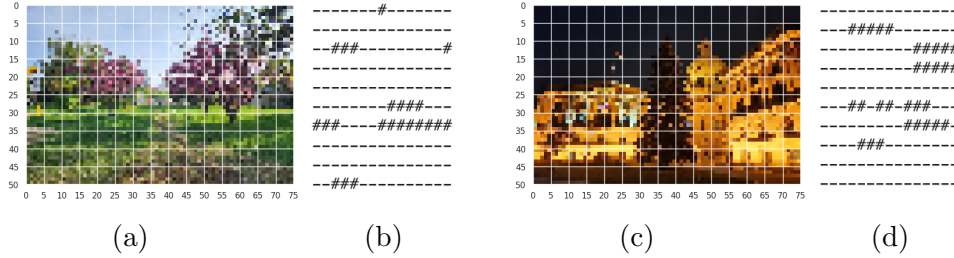


Figure 4.8: Examples of our model’s translation of natural images.

4.3.4 Case Studies on Translation Accuracy

Figure 4.9 represents four sample frames from the game Megaman. We have chosen this game to analyze our model’s performance on unseen games since it has very different aesthetics than our training games. Examples 4.9a and 4.9c were chosen from the top 30 samples with the best accuracy, and examples 4.9e and 4.9g were chosen from the set of 30 samples with the worst accuracy. The first two sample translations indicate that our model seems to be most accurate with frames that consist primarily of background tiles with even textures. On the other hand, the second two examples illuminate our model’s weakness in identifying background from foreground when frames have different textures and colors compared to the training data.

4.3.5 Case Studies on Translation of Natural Images

In this section, we provide two examples of natural images and our VAE-GAN’s translations of them in Figure 4.8. As expected, the translations don’t seem to meaningfully match the patterns in the inputs since these images are out-of-distribution. However, we can take these results as further evidence that our model isn’t suffering from posterior collapse.

Chapter 5

Conclusion

In this chapter, we examine the limitations inherent in our approach, explore potential avenues for future improvements, and draw final conclusions to complete this thesis.

5.0.1 Limitations and Future Work

In this thesis, we aimed to develop a framework that is able to perform level generation and translation. Our initial evaluations suggest that our VAE-GAN architecture was able to learn both of these tasks and achieve better results compared to other frequently used architectures in PCGML. However, there are multiple avenues for improvement. First, our initial experiment on the impact of the number of training games seems to suggest that more games may not improve our model’s generalizability over unseen games. However, expanding the training dataset may still boost the translation performance if we include a considerably larger number of games. Moreover, semi-supervised learning techniques can also provide a means to use the abundant but unannotated data available from gameplay videos [56]. Improved generalizability opens up the possibility of generating new level segments for unseen games. If the framework is able to translate unseen level segments effectively, one can take any gameplay video of a new 2D platformer game and feed its frames through the VAE to attain their translations. Then, these (frame, translation) pairs can be used to generate novel level segments for a new game.

Second, the evaluation results of the generated outputs of our model demon-

strate that there is significant room for improvement in terms of playability. Adding a new loss to the model to take playability into account or augmenting input data with additional information, such as player path, may lead to a better playability score for the generated outputs.

5.0.2 Conclusions

In this thesis, we aimed to answer the following pivotal research questions regarding the simultaneous learning of level generation and translation tasks.

- RQ1) Can learning level generation and translation tasks in tandem benefit any of these tasks, compared to learning each task separately?
- RQ2) To what extent can this idea improve or hinder the performance quality of the proposed architecture regarding both generation quality and translation accuracy?
- RQ3) What is a ML-based architecture that can be utilized to perform these tasks jointly?
- RQ4) How can such an approach potentially address the problem of low annotated data for procedural level generation via machine learning?

To tackle RQ3, we proposed a novel framework for simultaneous level segment generation and translation. We trained a novel VAE-GAN-based architecture on a dataset of human-annotated platformer games. To answer RQ1 and RQ2, we compared our framework against multiple baselines and showed evidence that learning these two tasks jointly can lead to an overall better performance in terms of generation quality and translation accuracy. Our experiments on generalizability provide inconclusive evidence of the potential ability of this approach to address the problem of insufficient data for PCGML, as stated in RQ4. Additional experiments with larger datasets containing data from more games may be helpful for further validating our findings and expanding the scope of applicability of our proposed method.

References

- [1] A. Amato, “Procedural content generation in the game industry,” *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*, pp. 15–25, 2017.
- [2] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International conference on machine learning*, PMLR, 2017, pp. 214–223.
- [3] S. Berns, V. Volz, L. Tokarchuk, S. Snodgrass, and C. Guckelsberger, “Not all the same: Understanding and informing similarity estimation in tile-based video games,” *arXiv preprint arXiv:2402.18728*, 2024.
- [4] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [5] A. Canossa and G. Smith, “Towards a procedural evaluation technique: Metrics for level design,” in *The 10th International Conference on the Foundations of Digital Games*, sn, 2015, p. 8.
- [6] E. Chen, C. Sydora, B. Burega, *et al.*, “Image-to-level: Generation and repair,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 189–195, Oct. 2020. DOI: 10.1609/aiide.v16i1.7429. [Online]. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/7429>.
- [7] F. Chollet *et al.*, *Keras*, <https://keras.io>, 2015.
- [8] B. De Kegel and M. Haahr, “Procedural puzzle generation: A survey,” *IEEE Transactions on Games*, vol. 12, no. 1, pp. 21–40, 2020. DOI: 10.1109/TG.2019.2917792.
- [9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [10] J. Feydy, T. Séjourné, F.-X. Vialard, S.-i. Amari, A. Trounev, and G. Peyré, “Interpolating between optimal transport and mmd using sinkhorn divergences,” in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 2681–2690.

- [11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” *Advances in neural information processing systems*, vol. 30, 2017.
- [13] M. Guzdial and M. Riedl, “Game level generation from gameplay videos,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 12, no. 1, pp. 44–50, Jun. 2021. DOI: 10.1609/aiide.v12i1.12861. [Online]. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/12861>.
- [14] M. Guzdial, S. Snodgrass, and A. J. Summerville, *Procedural Content Generation Via Machine Learning: An Overview*. Springer, 2022.
- [15] C. Harris, M. Stephens, *et al.*, “A combined corner and edge detector,” in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.
- [16] B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.
- [17] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, “A comparative evaluation of procedural level generators in the mario ai framework,” in *Foundations of Digital Games 2014, Ft. Lauderdale, Florida, USA (2014)*, Society for the Advancement of the Science of Digital Games, 2014, pp. 1–8.
- [18] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pmlr, 2015, pp. 448–456.
- [19] M. Jadhav and M. Guzdial, “Tile embedding: A general representation for level generation,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 17, no. 1, pp. 34–41, Oct. 2021. DOI: 10.1609/aiide.v17i1.18888. [Online]. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/18888>.
- [20] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017.
- [21] N. Y. Khameneh and M. Guzdial, “Entity embedding as game representation,” *arXiv preprint arXiv:2010.01685*, 2020.
- [22] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [23] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” in *International conference on machine learning*, PMLR, 2016, pp. 1558–1566.

- [24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [25] H. Liu, T.-H. Hong, M. Herman, T. Camus, and R. Chellappa, “Accuracy vs efficiency trade-offs in optical flow algorithms,” *Computer vision and image understanding*, vol. 72, no. 3, pp. 271–286, 1998.
- [26] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3730–3738.
- [27] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *IJCAI’81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.
- [28] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, “Are gans created equal? a large-scale study,” *Advances in neural information processing systems*, vol. 31, 2018.
- [29] L. van der Maaten and G. E. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: <https://api.semanticscholar.org/CorpusID:5855042>.
- [30] J. Mariño, W. Reis, and L. Lelis, “An empirical evaluation of evaluation metrics of procedurally generated mario levels,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 11, 2015, pp. 44–50.
- [31] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [32] M. L. Rizzo and G. J. Székely, “Energy distance,” *wiley interdisciplinary reviews: Computational statistics*, vol. 8, no. 1, pp. 27–38, 2016.
- [33] M. Rosenblatt, “Remarks on some nonparametric estimates of a density function,” *The annals of mathematical statistics*, pp. 832–837, 1956.
- [34] Y. Rubner, C. Tomasi, and L. J. Guibas, “A metric for distributions with applications to image databases,” in *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, IEEE, 1998, pp. 59–66.
- [35] D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations*. MIT press, 1986.
- [36] A. Sarkar and S. Cooper, “Dungeon and platformer level blending and generation using conditional vaes,” in *2021 IEEE Conference on Games (CoG)*, IEEE, 2021, pp. 1–8.

- [37] 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*, 2021, pp. 1–11.
- [38] 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, 2020, pp. 280–286.
- [39] A. Sarkar, Z. Yang, and S. Cooper, “Controllable level blending between games using variational autoencoders,” *arXiv preprint arXiv:2002.11869*, 2020.
- [40] N. Shaker, J. Togelius, and M. J. Nelson, “Procedural content generation in games,” 2016.
- [41] J. Shi *et al.*, “Good features to track,” in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, IEEE, 1994, pp. 593–600.
- [42] D. Smirnov, M. GHARBI, M. Fisher, V. Guizilini, A. Efros, and J. M. Solomon, “Marionette: Self-supervised sprite learning,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 5494–5505. [Online]. Available: <https://proceedings.neurips.cc/paper/2021/file/2bcab9d935d219641434683dd9d18a03-Paper.pdf>.
- [43] A. M. Smith and M. Mateas, “Answer set programming for procedural content generation: A design space approach,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011.
- [44] G. Smith, “An analog history of procedural content generation.,” in *FDG*, Boston, MA, 2015.
- [45] G. Smith, “Procedural content generation: An overview,” *Level Design*, pp. 159–183, 2017.
- [46] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, “Rhythm-based level generation for 2d platformers,” in *Proceedings of the 4th international Conference on Foundations of Digital Games*, 2009, pp. 175–182.
- [47] S. Snodgrass and S. Ontanon, “An approach to domain transfer in procedural content generation of two-dimensional videogame levels,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 12, 2016, pp. 79–85.
- [48] S. Snodgrass and S. Ontanón, “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, 2016.

- [49] S. Snodgrass, A. Summerville, and S. Ontanon, “Studying the effects of training data on machine learning-based procedural content generation,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, no. 1, pp. 122–128, Jun. 2021. DOI: 10.1609/aiide.v13i1.12930. [Online]. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/12930>.
- [50] A. Summerville, “Expanding expressive range: Evaluation methodologies for procedural content generation,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 14, no. 1, pp. 116–122, Sep. 2018. DOI: 10.1609/aiide.v14i1.13012. [Online]. Available: <https://ojs.aaai.org/index.php/AIIDE/article/view/13012>.
- [51] A. Summerville and M. Mateas, “Super mario as a string: Platformer level generation via lstms,” *arXiv preprint arXiv:1603.00930*, 2016.
- [52] A. Summerville, S. Snodgrass, M. Guzdial, *et al.*, “Procedural content generation via machine learning (pcgml),” *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.
- [53] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontanón, “The vglc: The video game level corpus,” *arXiv preprint arXiv:1606.07487*, 2016.
- [54] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [55] 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.
- [56] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.
- [57] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.
- [58] F. Wilcoxon, “Individual comparisons by ranking methods,” in *Breakthroughs in statistics: Methodology and distribution*, Springer, 1992, pp. 196–202.