Analysing the effect of latent space mutation strategies for PCGML

Kayleize Govender, Branden Ingram, Pravesh Ranchod
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
kayleize.govender@students.wits.ac.za
{branden.ingram, pravesh.ranchod}@wits.ac.za

***Abstract*—Procedural Content Generation Via Machine Learning (PCGML), describes an evolving area of research, that incorporates the use of machine learning models in a generative context. These innovative approaches accelerate the creation of content while concurrently reducing the need for human intervention. In particular, autoencoder models have been steadily employed in collaboration with supporting algorithms to facilitate the generation of novel game content. The standard autoencoder provides a means of replicating given data by learning a lower dimensional embedding, called the latent representation. Applying various operations to these representations can yield uniquely diverse content. This paper offers a versatile and robust framework for game level generation using a standard autoencoder for level generation and a denoising autoencoder for level repair and enhancement. Additionally, the integration of clustering techniques effectively identifies the core components that make up the various types of levels in a Rogue-like game domain. The essence of this approach lies in harnessing the knowledge within the clusters to guide the mutation phase of the level's latent spaces. This refined methodology provides significant insights on level analysis, generation and repair, highlighting how autoencoders can be used as a basis for game development.**

***Index Terms*—PCGML, Supervised Learning, Level Generation**

I. Introduction

Content generation in the field of game-level development has been a topic of interest to academic researchers for over a decade and is made up of many varied techniques [1]. These approaches are becoming increasingly crucial in modern video games due to their ability to enhance scalability, increase diversity, and offer personalization, which improves player engagement at a more affordable cost [2], [3]. Procedural Content Generation via Machine Learning (PCGML) represents one of these methodologies, which involves the training of models on pre-existing, human-curated datasets of game levels. It is this learned model which is then utilised to generate new levels that exhibit analogous characteristics [4].

While this method still necessitates the involvement of human designers, it proves significantly more intuitive compared to search-based PCG methods or constraint-based approaches. These latter methods demand the development of fitness/reward or constraint functions, respectively, to direct the generation process. Such functions often result in unforeseen outcomes [5].

The field of PCGML is not limited to generation, but the methods of modelling content can be extended to the areas of compression, repair, recognition and analysis [4]. Many different techniques have been explored for level generation, such as neural networks [4], Markov models [6], Long Short-Term Memory Recurrent Neural Networks as shown by Summerville and Mateas (2016) [7], as well as clustering [4]. The task of game-level generation is becoming increasingly complex. Today, game levels can span a large state space, making the process of obtaining smaller and more compact representations a necessity. The natural ability of an autoencoder is dimensionality reduction. Combining this strength with intuitive algorithms forms the foundation for level generation through autoencoders. A frequently employed variant of the autoencoder is the variational autoencoder (VAE), renowned for its generative capabilities. VAEs have played a key role in controllable level blending, as demonstrated in the works of [8], [9], and [10]. Past research has notably focused on optimizing the reconstruction loss of the autoencoder to promote learning of a "good" latent space embedding such that "good" levels can be sampled from this space [9].

In this paper, we propose different techniques for sampling from a learned latent space, which we call "Latent Space Mutation Strategies". We investigate the effect of traditional strategies which incorporate adding noise as well as novel clustering-based mutation strategies. Clustering has long been a useful technique in identifying patterns within a dataset, and it is this feature that we look to leverage when sampling from the latent space [11], [12]. Rather than simply adding noise can you mutate a latent space vector based on some identified patterns. To this end, we train a VAE autoencoder model which is connected to a denoising autoencoder to repair generated levels that suffer from "incompleteness" as in [13]. This model is trained on a pre-existing dataset of levels generated in a 2.5D rogue-like grid environment. After this, we sample novel latent space vectors and process them through our latent space mutators. These mutated vectors are then processed through

979-8-3503-5067-8/24/$31.00 ©2024 IEEE

the rest of the trained model to output a final level. The generated and repaired levels are assigned a novelty score measuring the similarity in comparison with the original levels, to establish how effective the different mutation strategies are in generating unique content.

II. Related Work

The potential to reduce development costs, increase replayability and promote diverse gameplay experiences has led to a significant amount of attention from both academia [14]–[16] and commercially [17]–[21]. This has resulted in a number of approaches namely; search-based [22], [23], constraint-based [24], [25] and data-based algorithms [4], [13].

A. General PCG

Approaches which utilise a fitness or reward function in an iterative training regime are usually considered to be search-based approaches. However, this search process tends to be computationally expensive and the design of the objective function can be unintuitive [26]. Constraint-based approaches rely more heavily on designed elements in the form of rules which govern legal actions in a generation pipeline. In grid-based domains, neighbourhood rules for cellular automata have been used to generate cave systems [27]. Alternatively, generative grammars utilised linguistic rules and formal language representations to generate replacement rules [28]. This concept was expanded to graph structures to generate dungeons where nodes represented rooms and edges its adjacencies [29]. These approaches all suffer from enforcing global constraints due to rules only enforcing local changes. Hierarchical approaches have been applied to circumvent issues related to this limitation [24], however, in doing so incurs a greater design cost.

B. PCGML

Procedural content generation via machine learning (PCGML) looks to avoid the need to construct appropriate reward/fitness functions nor does it require expert knowledge for the purpose of designing constraints [4]. Alternatively, PCGML utilises large data sets of quality example content to train a model to generate levels in a supervised setting. Jain et al. (2016) constructed denoising autoencoders to overcome the detriment of over-fitting when using a standard autoencoder [13]. In this context, the autoencoder was trained on input that contained random noise, while the original input was used as the target data. Thakkar et al. (2019) extend this by using a multi-channel encoding along with a variational autoencoder and compared the results to the standard autoencoder [10]. The generation takes place in two phases, using the autoencoder twice. The first phase uses the autoencoder to create a gene pool of 10,000 samples, by supplying random seeds into the decoder. The next phase evolved the levels by applying an evolutionary process. The evolutionary process first consisted of applying the crossover and mutation function. The idea was to combine two images by assigning different proportions of each to a child image (crossover), then randomly modifying its latent variable in the latent space of the autoencoder (mutation). More recently, variational autoencoders have been used to blend two existing levels [8]. Here Sarkar et al. (2020) trained the autoencoder on encoded window segments from each level instead of the entire level [8]. Here the previous window acted as a prior to influence the generation of the neighbouring window. Similarly, our clusters represent a prior influencing the generation of a stand-alone level. Alternatively, González-Duque et al. (2022) utilised a differential geometry approach for sampling from a learned latent space in Mario to bias towards playable levels [30].

The above approaches highlight the main strategies that influence the following generation methods.

III. Autoencoders

The autoencoder model is based on the neural network architecture. Many types of autoencoders differ mainly in their composition and the process by which they learn to encode and decode the given data. Standard autoencoders learn to replicate the data it has been fed, while producing a lower representation that holds its underlying characteristics [31]. Obtaining these representations allows for the application of additional techniques and more explicitly, the use of mathematical operations, to alter the original representations and in turn create modified versions of them [13]. Contrasting this technique to traditional approaches in which most, if not the entire content space was examined and modified, provides a more efficient way of "manipulating" existing levels to create new levels. The additional benefit of a trained autoencoder is that it can be decomposed into two independent parts. The encoder is responsible for creating the lower dimensional code, which resides in the hidden layer, connecting the encoder and decoder. The decoder is then tasked with transforming this lower-level data back into its original state. Once the model has been trained, the decoder can be queried by feeding it unseen vectors that match the dimension of the hidden layer. A diagram showing how new levels can be generated is shown in Figure 1

IV. Methodology

Our method uses both standard and clustering-based generation techniques alongside the usage of an autoencoder architecture to produce video game levels.

A. Generation Process

To generate levels we train two separate variational autoencoder models. The first is trained to replicate input levels from our level dataset and is called the "Generator". The second is trained on corrupted versions of these levels and serves to repair this corruption from the level and is called the "Denoiser". This process of level repair incorporates a variant of the denoising autoencoder used in [13], where the aim was to add random noise to the training data. The decoder from the "Generator" is combined with the "Denoiser" to form the online generation pipeline. This pipeline is fed novel inputs which are generated using multiple differing latent space mutation strategies to generate novel levels.

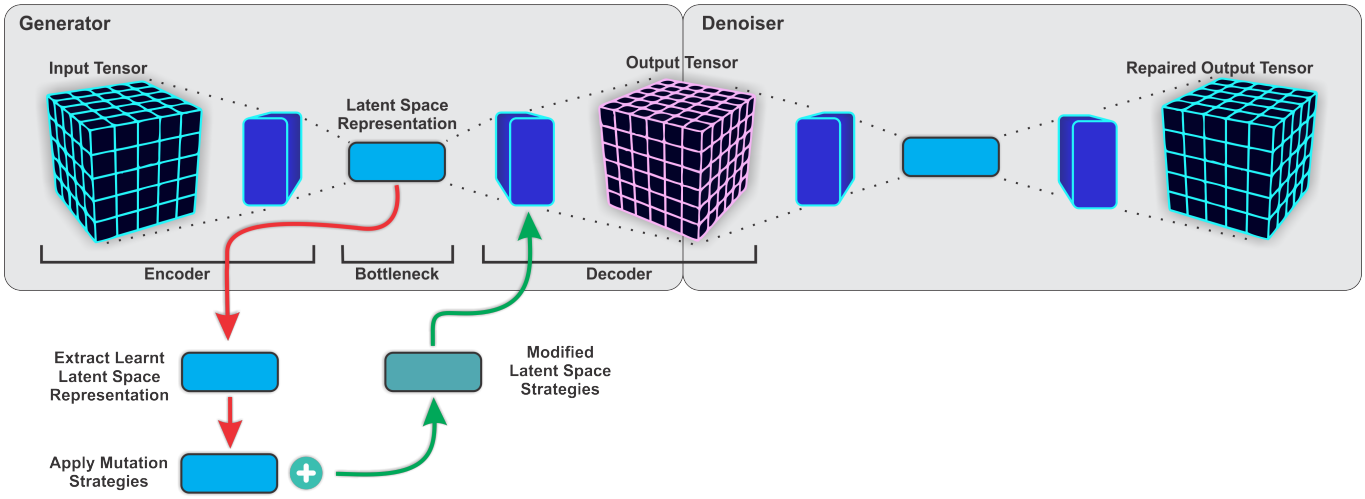


Fig. 1: Generative pipeline, corresponding both training and inference modes. During training a level is converted to a tensor which is then reduced to a lower dimensional latent space vector. This vector is then projected back up to the original size, the difference between these forming the "Generator" reconstruction loss. The "Denoiser" is trained separately on corrupt versions of the original dataset. During Inference both generators are combined we then sample an extracted latent space representation from the "Generator" form which we apply a mutation strategy. This mutated representation is then passed through the rest of the network producing a final level.

B. Latent Space Mutation Strategies

Building on past work of adding random noise and blending levels, the following strategies are used to generate new game levels.

1) ***Random Uniform Noise****:* Randomly sampling a vector from a uniform distribution to pass through our pipeline.

2) ***Normal Distribution Sampling****:* Randomly sampling each component of a vector from a normal distribution using the mean and standard deviation from the corresponding dimension of the learnt latent representations of the original game levels:

$$\mathbf{Z} = (N_1(\mu_1, \sigma_1), N_2(\mu_2, \sigma_2), \ldots, N_n(\mu_n, \sigma_n)). \quad (1)$$

Here n represents the size of the latent vector and N_i represents a normal distribution defined by μ_i and σ_i the mean and variance corresponding to a particular feature i.

3) ***Addition Of Noise To Latent Vectors****:* Extracting the learnt latent representations and adding a uniform noise vector with the same dimension.

4) ***Random Latent Interpolation****:* Applying the Interpolation formula between two existing, randomly selected latent vectors L_1 and L_2 :

$$\mathbf{L}_{\text{blended}} = \alpha\mathbf{L_1} + (1-\alpha)\mathbf{L_2}, \text{where } \alpha \in [0, 1]. \quad (2)$$

The interpolation formula is applied to all cluster-based methods with α = 0.5.

5) ***Same Cluster****:* Randomly Select two latent representations from the same cluster and interpolate between them.

6) ***Closest Clusters****:* Randomly choose and interpolate between two latent representations from clusters with centroids that are nearby.

7) ***Furthest Clusters****:* Randomly choose and interpolate between two latent representations from clusters with centroids that are maximally distant.

8) ***Centroids****:* In addition to facilitating appropriate level groupings, the clusters also feature centroids representing the average of all latent representations within a cluster. These centroids trivially capture the shared characteristics of the latent vectors within their respective clusters.

V. Experiments

A. Rogue-Like Dataset

848 Game levels represented as 2-D arrays are extracted from a collection of text files. These levels were generated by [24] and are similar to games used in prior works, such as *The Legend of Zelda* [22]. The arrays consist of 28×28 tokens, where each token value corresponds to a specific colour indicating the type of tile it is, e.g.' 0' represents green which symbolizes grass. The files include values from 0-9, representing 10 different colours/classes of tiles. All levels are one-hot-encoded to create a $28 \times 28 \times 10$, 3-D tensor.

B. Data-Corruption

Corruption refers to selecting non-green tiles at random and assigning them the integer 0, representing green tiles, shown in Figure 2. This is to overcome the incomplete characteristics of levels generated by the "Generator".

C. Training

The "Generator" model is a 2-layer, fully connected encoder and decoder using ReLU and softmax activations respectively. The encoder transforms the flattened input 3-D tensor ($28 \times 28 \times 10$) into a vector of 392 dimensions, as seen in Figure

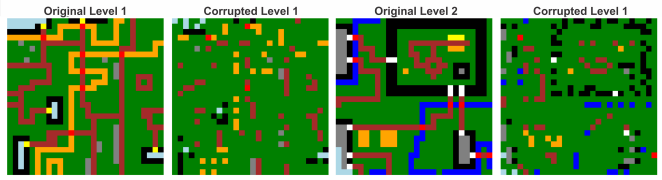


Fig. 2: Depiction of two levels with their corresponding versions after apply 50% corruption.

1. The decoder then converts this 392 dimension vector back into the 7840 vector. The output of the decoder is reshaped to a 3-D tensor of shape $28 \times 28 \times 10$, where the "argmax" is taken and the final 2-D array of 28×28 representing the level, is obtained. The model is trained using the Adam Optimizer with a learning rate of 0.001 and the *binary cross-entropy* function to compute the training loss [8]. Given the goal is to generate new levels by constructing unseen vectors, the model is trained, validated and tested with a split ratio of $60 : 20 : 20$ respectively. The training and validation loss curves can be seen in Figure 3. The mean similarity of the replicated levels in the test data is approximately 67%, showing that the standard autoencoder managed to capture a decent proportion of the underlying level characteristics. To extract the latent representations for analysis, mutation and blending, the model is retrained on the entire dataset in batches of 256 for a total of 100 epochs.

Using a split ratio of $80 : 20$ for training and testing data, the denoising autoencoder was evaluated on different corruption percentages as seen in Table I. The mean similarity to the original levels is computed, to evaluate the model's ability to reconstruct unseen corrupt levels. A corruption percentage of 50% is chosen for the final denoising autoencoder.

Corruption %	Mean Similarity %
30	66.06
40	64.98
50	**63.18**
60	63.11
70	61.92
80	60.67

TABLE I: Reconstructive similarity score based upon the degree of level corruption.

D. Clustering

For the clustering component of the model, we utilised k-means clustering. To choose the appropriate number of clusters, different values of k were initially selected, however, we later settled for 6 based on visualized tile distributions. To visualize the clusters, the latent representations were projected into two-dimensional space using *Principal Component Analysis (PCA)*. For k-means, we fit our data using 100 restarts and a maximum iteration of 10000. In addition, k-means used a tolerance of 0.0001

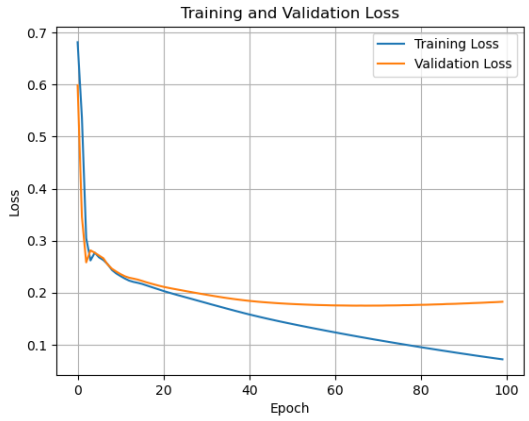


Fig. 3: Binary cross entropy loss curves for "Generator" during Level replication training.

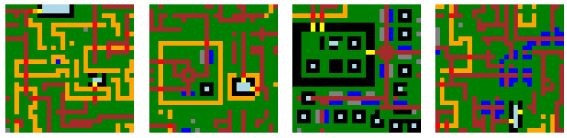


Fig. 4: Levels generated using just the "Generator" part of the pipeline depicting artefacts of "incompleteness".

E. Performance Metric

Measuring the unique properties of the generated levels is done using mean similarity. *Mean Similarity* is calculated by comparing each generated level to all 848 existing levels and computing the sum of equal tiles which is then averaged across the number of generated levels. A common goal similarity would lie between 55% and 75%, illustrating that the levels portray characteristics of the data used to create it, but also incorporate additional qualities that would describe it to be unique.

VI. Results and Discussion

This section presents our results from the standard offline training process as well as the effects of the differing mutation strategies.

A. Autoencoder Training Performance

In Figure 3 we observe that our model replicates all the levels with 96% accuracy when measured using a Binary Cross Entropy Loss but few of the structural entities were "incomplete", as seen in Figure 4 where roads, walls and rivers are not connected. Additionally, we observe that after "epoch 40" the validation loss converges while the training loss continues to decline. This is indicative of the model beginning to overfit to the training data. One approach to mitigate this would be to incorporate early stopping, however, we opted not to, since we are focused on measuring the impact of the mutation strategies and not optimising the model's reconstruction performance.

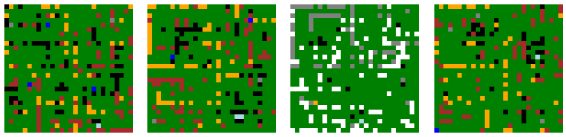


Fig. 5: Reconstructed levels generated from passing through randomly generated noisy latent space vectors through the Decoder portion of the "Generator".

Figure 5 shows randomly sampled latent vectors from a uniform distribution which are reconstructed by passing through the standard decoder. The generated levels show similar traits of incompleteness as the corrupted levels in Figure 2, with partially observable entities and paths. This can be identified as an over-fitting property of the "Generator".

The design of the "Generator" to output a probability distribution of all tiles paired with the property of the original game levels that contain approximately 50% of green tiles, confirms that the autoencoder most likely assigns that label "0" in the case of uncertainty. Although the bottleneck sizes for each autoencoder are the same, the encoder of the "Denoiser" is used to obtain a suitable latent representation that the decoder is familiar with. Figure 6a illustrates the capabilities of the "Denoiser" on a partially generated level from uniform random noise, obtained using the "Generator".

B. Non-Cluster Based Latent Mutation

Figures 6a, 6b, 6c and 6d, were generated by selecting two levels from a set of 20 levels generated for each strategy mentioned above. The example levels were obtained using the decoder portion of the "Generator" for generation and the "Denoiser" for repair.

Generation Strategy	Mean Similarity %	Average Green Tile Density %
Uniform Noise	47.26	81.75
Sample Normal Distribution (mean,std)	40.46	62.48
Existing Latent + Uniform Noise	47.11	81.48
Random Interpolation	39.30	61.94

TABLE II: Evaluation Of non-cluster based level generation strategies in terms of similarity to the original dataset and density of green tiles.

Observing Table II and Figures 6a, 6b, 6c, 6d, our model shows excellent recall in completing enclosure entities although struggles with forming coherent paths. Among the 4 non-cluster-based strategies, sampling from a normal distribution and interpolating random latent vectors showed the lowest mean similarity to the original levels. Although the similarity percentage falls below 50%, the average tile density falls within a descent range to the cluster proportions. Producing more than 20 generated levels may result in a more diverse set of levels that incorporate many of the existing features. However, the results for non-cluster-based generation showcase that basic mathematical operations define a natural starting point for latent mutation.

(a) Random Uniform Noise.

(b) Addition Of Noise To Latent Vectors.

(c) Normal Distribution Sampling

(d) Random Latent Interpolation.

Fig. 6: Combined Results for level generation using complete pipeline when using non-cluster based latent mutation strategies.

C. Cluster Based Latent Mutation

Clustering techniques demonstrate how the different levels can be grouped together. Intuitively, this is a way to attain more knowledge of the features that make up the levels. Exploiting this knowledge allows for a controllable mutation process over the level's latent space. Figure 7 showcases the centroids for each cluster generated using the standard decoder. Visualizing these centroids suggests that levels are grouped based on the presence of rectangular or square enclosures. However, since these enclosures only constitute a portion of a complete level, the latent vectors closest to the centroids in terms of the shortest Euclidean distance, as depicted in Figure 8, are selected as the "chosen" centroids for blending. The corresponding tile distributions for each cluster can be seen in Figure 10. The key observation in the tile proportions is that in all clusters the proportion of green tiles is approximately 50%. These distributions serve as a baseline for comparison with the generated levels.

To use the "closest centroids" strategy we calculated the pairs of the closest centroids, as listed in Table III, to be $[(1,5),(1,6),(2,6),(5,6)]$. Similarly, we calculated the pairs of the furthest centroids, as listed in Table III, to be

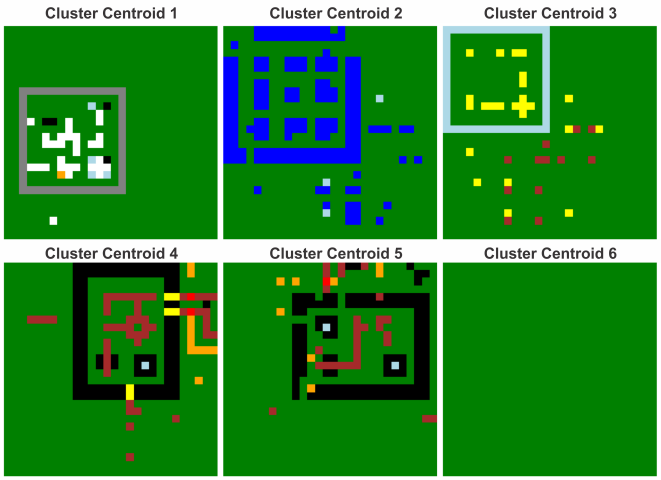


Fig. 7: Levels generated when passing latent centroid vector as input through the whole pipeline starting at the Decoder of the "Generator" network.

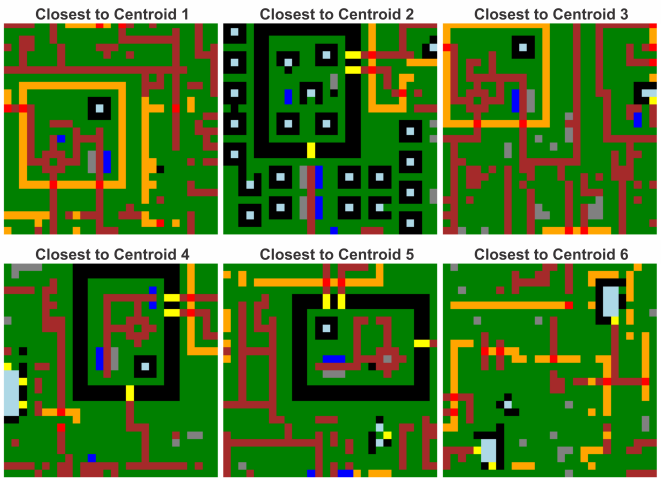


Fig. 8: Levels generated when passing the latent vector from the original dataset which is closest to the corresponding centroid vector as input through the whole pipeline starting at the Decoder of the "Generator" network.

$[(1,3),(2,3),(3,5),(3,6)]$.

Figures 11a, 11b, 11c and 12, were blended using $\alpha = 0.5$, allowing equal proportions of each level to contribute to the blended level. Figure 11b and 11c show examples of levels obtained from using the closest and furthest centroid pairs respectively. 5 levels were generated For each of the 4 pairs, by randomly selecting latent vectors in the relevant cluster pairings. Figure 11b shows examples of levels obtained from using the latent vectors closest to the respective centroid. Each of these vectors was interpolated with every other centroid vector, resulting in a set of 15 generated levels.

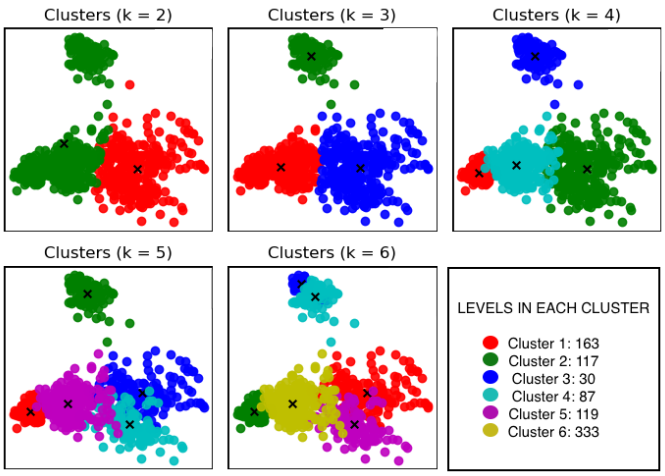


Fig. 9: K-Means clusters of latent representations of all original levels in 2-D space for $k \in [2,6]$ and corresponding number of levels in each cluster for $k = 6$.

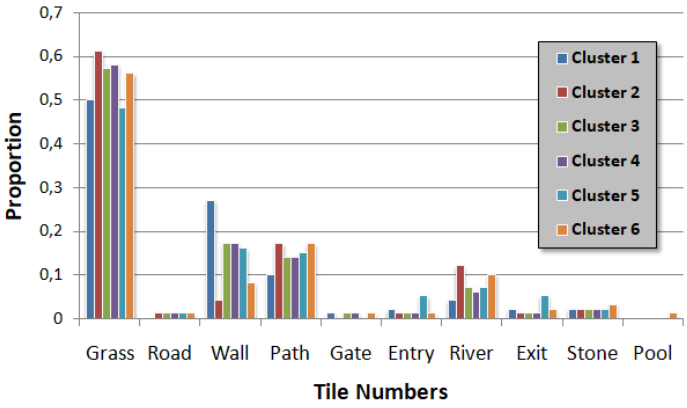


Fig. 10: Tile distributions of original levels in each cluster.

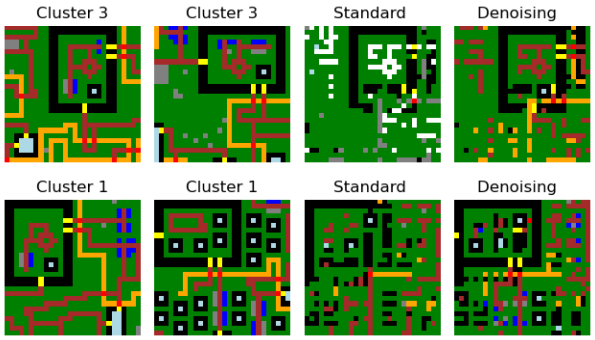
Blended Generation Strategy	Mean Similarity %	Average Green Tile Density %
Same Cluster ($\alpha = 0.5$)	38.54	60.52
Closest Cluster	38.52	59.69
Furthest Cluster	37.76	56.35
Centroids	40.62	67.35

TABLE IV: Evaluation Of cluster based level generation strategies in terms of similarity to the original dataset and density of green tiles.

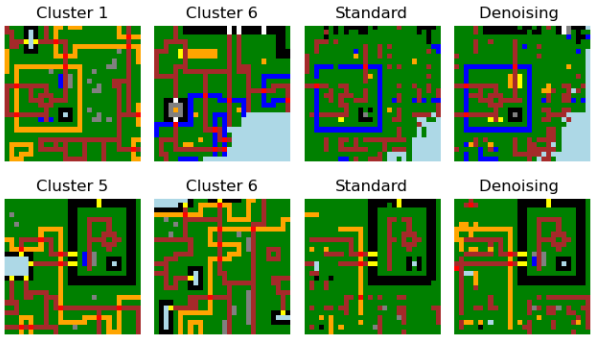
Table IV shows the mean similarity and green tile density for the cluster-based methods. Overall, the similarity percentages are much lower than that of the non-cluster-based strategies. The example Figures 11a, 11b, 11c and 12, showcase more diverse range levels among the 4 strategies. Observing the tile proportion plots in Figure 13, the green tile density closely resembles the original density for the cluster tile proportions in Figure 10. The proportion of blue, black and maroon tiles, representing rivers, walls and paths, are lower than the cluster proportions, but the wall and path proportions appear to be evenly distributed among the generated levels. The use of alternate clustering strategies could provide more

Centroids	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Cluster 1	-	65.42	**83.76**	62.71	48.65	46.6
Cluster 2	-	-	**86.34**	69.91	65.04	45.78
Cluster 3	-	-	-	71.71	**87.53**	76.84
Cluster 4	-	-	-	-	67.51	56.78
Cluster 5	-	-	-	-	-	46.36
Cluster 6	-	-	-	-	-	-

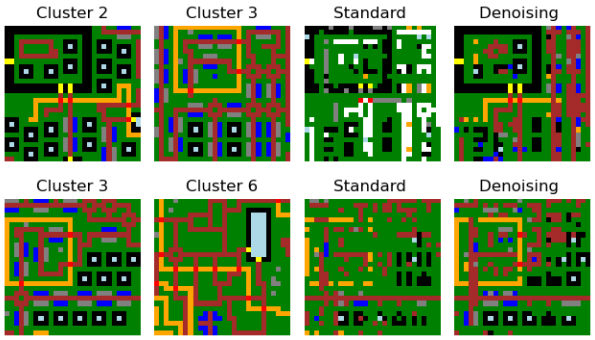
TABLE III: Euclidean distances between cluster centroids.



(a) Same Cluster



(b) Closest Clusters



(c) Furthest Clusters

Fig. 11: Combined Results for level generation using complete pipeline when using cluster based latent mutation strategies.

refined groupings.

VII. Future Work

The interpolation formula for all blending strategies uses an equal blending proportion for each level, this could be optimized by allocating a larger proportion of the blending ratio to the more dense level or vice-versa, allowing for less stochasticity during blending. The simultaneous lack of

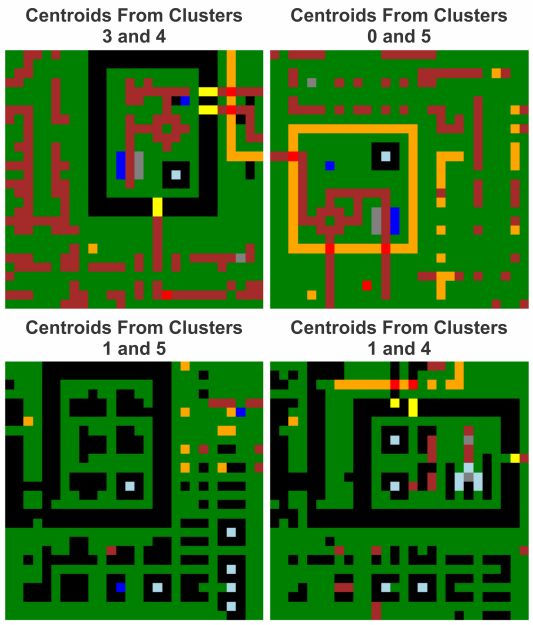


Fig. 12: Centroids.

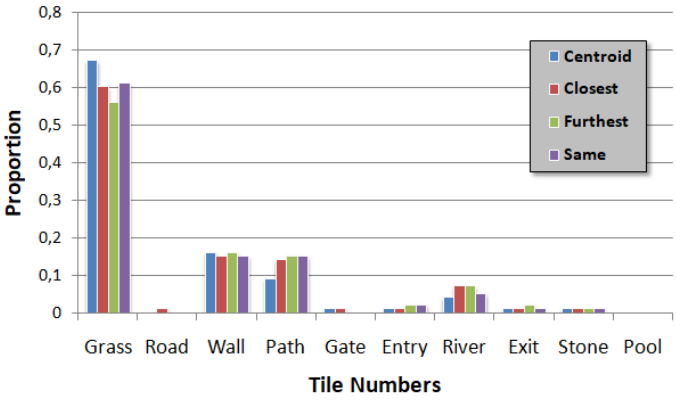


Fig. 13: Average tile proportions for each blending strategy.

diversity and similarity can be attributed to numerous design decisions within the frame. The simple structure of the autoencoders provides a straightforward implementation and generation process, however results in less authentic levels. The use of more complex architectures could improve the amount of complex information retained by the autoencoder. The results for the various strategies showcase the need for techniques that make use of the information residing in the latent representations. Although considered a trivial way of mutation, the simple addition of noise or vectors disregards their contents. Using search or fitness functions to evolve the latent vectors would incorporate more of the learned patterns leading to novel insights.

VIII. Conclusion

The work has presented a comprehensive framework for game-level generation using autoencoders and clustering techniques by highlighting their capabilities and limitations. The

pipeline of level replication, analysis and repair can be adapted and transformed for various use cases. Although the generated levels were not up to standard under the metric criteria, the visual examples exhibit the model's ability to introduce variations to existing levels. In general, autoencoders are not the trivial choice for level generation but have proven to be a sufficient foundation for further complex techniques. Future areas of exploration include generating levels with set objectives and rules, to examine how the autoencoder can handle generating content under given constraints. The integration of more sophisticated mutation strategies, such as genetic algorithms, incorporates the vital elements captured in the latent representations.

REFERENCES

[1] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, feb 2013. [Online]. Available: https://doi.org/10.1145/2422956.2422957

[2] G. Smith, "Procedural content generation: An overview," *Level Design Processes and Experiences*, pp. 159–183, 2017.

[3] O. Korn, M. Blatz, A. Rees, J. Schaal, V. Schwind, and D. Görlich, "Procedural content generation for game props? A study on the effects on user experience," *Comput. Entertain.*, vol. 15, no. 2, pp. 1:1–1:15, 2017. [Online]. Available: https://doi.org/10.1145/2974026

[4] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. 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.

[5] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural Content Generation: Goals, Challenges and Actionable Steps," in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 61–75. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4336

[6] S. Snodgrass, *Markov models for procedural content generation*. Drexel University, 2018.

[7] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," *arXiv preprint arXiv:1603.00930*, 2016.

[8] A. Sarkar, Z. Yang, and S. Cooper, "Controllable level blending between games using variational autoencoders," *arXiv preprint arXiv:2002.11869*, 2020.

[9] A. Sarkar and S. Cooper, "Sequential segment-based level generation and blending using variational autoencoders," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020, pp. 1–9.

[10] S. Thakkar, C. Cao, L. Wang, T. Choi, and J. Togelius, "Autoencoder and evolutionary algorithm for level generation in lode runner," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–4.

[11] B. Ingram, C. van Alten, R. Klein, and B. Rosman, "Generating interpretable play-style descriptions through deep unsupervised clustering of trajectories," *IEEE Transactions on Games*, 2023.

[12] M. Guzdial and M. Riedl, "Game level generation from gameplay videos," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 12, no. 1, 2016, pp. 44–50.

[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*, vol. 9, 2016.

[14] J. Dormans, "Adventures in level design: generating missions and spaces for action adventure games," in *Workshop on procedural content generation in games*, 2010.

[15] J. Dormans and S. Leijnen, "Combinatorial and exploratory creativity in procedural content generation," in *Workshop on Procedural Content Generation for Games*, 2013.

[16] 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 Trans. Games*, vol. 10, no. 3, pp. 257–270, 2018. [Online]. Available: https://doi.org/10.1109/TG.2018.2846639

[17] T. Adams and Z. Adams, "Dwarf fortress," http://www.bay12games.com/dwarves/, 2006, accessed: 2023-02-01.

[18] D. Yu, "Spelunky," https://spelunkyworld.com/, 2008, accessed: 2023-02-01.

[19] J. Dormans, "Cyclic generation," in *Procedural Generation in Game Design*. AK Peters/CRC Press, 2017, pp. 83–96.

[20] Ludomotion, "Unexplored," https://store.steampowered.com/app/506870/Unexplored/, 2017, accessed: 2023-02-01.

[21] T. Adams, "Emergent narrative in dwarf fortress," in *Procedural storytelling in game design*. AK Peters/CRC Press, 2019, pp. 149–158.

[22] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 95–101.

[23] Z. Jiang, S. Earle, M. Green, and J. Togelius, "Learning controllable 3d level generators," in *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022, pp. 1–9.

[24] M. Beukman, B. Ingram, I. Liu, and B. Rosman, "Hierarchical wavefunction collapse," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, no. 1, 2023, pp. 23–33.

[25] 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.

[26] D. Corus, D.-C. Dang, A. V. Eremeev, and P. K. Lehre, "Level-based analysis of genetic algorithms and other search processes," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 707–719, 2017.

[27] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 2010, pp. 1–4.

[28] N. Chomsky, *Language and mind*. Cambridge University Press, 2006.

[29] D. Adams *et al.*, "Automatic generation of dungeons for computer games," *Bachelor thesis, University of Sheffield, UK. DOI= http://www.dcs. shef. ac. uk/intranet/teaching/projects/archive/ug2002/pdf/u9da. pdf*, 2002.

[30] M. González-Duque, R. B. Palm, S. Hauberg, and S. Risi, "Mario plays on a manifold: Generating functional content in latent space through differential geometry," in *2022 IEEE Conference on Games (CoG)*. IEEE, 2022, pp. 385–392.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1st ed. MIT Press, 2016, ch. 14, pp. 502–525. [Online]. Available: http://www.deeplearningbook.org