Sequential Segment-based Level Generation and Blending using Variational Autoencoders

Anurag Sarkar
Northeastern University
sarkar.an@northeastern.edu

Seth Cooper
Northeastern University
se.cooper@northeastern.edu

ABSTRACT

Existing methods of level generation using latent variable models such as VAEs and GANs do so in segments and produce the final level by stitching these separately generated segments together. In this paper, we build on these methods by training VAEs to learn a sequential model of segment generation such that generated segments logically follow from prior segments. By further combining the VAE with a classifier that determines whether to place the generated segment to the top, bottom, left or right of the previous segment, we obtain a pipeline that enables the generation of arbitrarily long levels that progress in any of these four directions and are composed of segments that logically follow one another. In addition to generating more coherent levels of non-fixed length, this method also enables implicit blending of levels from separate games that do not have similar orientation. We demonstrate our approach using levels from *Super Mario Bros.*, *Kid Icarus* and *Mega Man*, showing that our method produces levels that are more coherent than previous latent variable-based approaches and are capable of blending levels across games.

CCS CONCEPTS

• **Applied computing** → *Computer games.*

KEYWORDS

procedural content generation, variational autoencoder, level generation, level blending, game blending, PCGML

ACM Reference Format:
Anurag Sarkar and Seth Cooper. 2020. Sequential Segment-based Level Generation and Blending using Variational Autoencoders. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3402942.3409604

1 INTRODUCTION

Procedural content generation via machine learning (PCGML) [30] refers to a subset of PCG techniques that use ML models for producing content. Several recent works [16, 23, 32, 33] in this field have made use of latent variable models such as Generative Adversarial Networks (GANs) [7] and Variational Autoencoders (VAEs) [13] for generating levels for platformers as well as dungeon crawlers. These models learn a continuous, latent representation of input game levels that can then be used to generate new levels via sampling, interpolating between points in this latent space, as well as by evolving latent vectors based on a given objective in order to generate levels with desired properties. While effective, these models work with fixed-size inputs and outputs, which limits the scope and coherence of levels that can be generated. Prior works produce levels by generating segments of levels independently and then stitching them together one after another. Though this can be satisfactory for platformers that proceed along a single direction or dungeon crawlers where gameplay can take place in discrete rooms, this approach would fail to generate playable, coherent levels for games where levels can proceed along multiple directions, both horizontally and vertically. Moreover, even for unidirectional platformers like *Super Mario Bros.*, such methods are not ideal since randomly sampling successive level segments does not ensure that the current segment logically follows from previous ones.

In this work, we address these issues via a simple modification to the training procedure for such models, combined with the use of a classifier for segment placement. More specifically, we train VAEs on game level segments, but rather than training the model to reconstruct the current input segment, as is the norm, we train it to reconstruct the segment that follows it in the input game level. That is, decoding a segment's encoded latent representation yields not the segment itself but the segment immediately after it in the level's progression. Thus, starting from an initial segment, this approach enables the generation of arbitrarily long levels composed of segments that logically follow from one to the next, via an iterative loop of encoding and decoding. In addition to this modification, we also train a random forest classifier to determine whether the next segment should be placed above, below, to the left or to the right of the current segment. This VAE-classifier combination thus enables us to generate continuous, coherent platformer levels of desired length, progressing in multiple directions, while still working with fixed-sized segments. Moreover, such a model also enables the generation of not just segments that blend together multiple games as in prior work [23], but entire blended levels without having to perform turn-based generation using multiple models as in [22].

We demonstrate our approach for three different platformer games—*Super Mario Bros.*, *Kid Icarus*, and *Mega Man*—as well as for the blended *Super Mario Bros.–Kid Icarus* domain. The contribution of this work is a new generative approach that enables:

(1) generation of more coherent platformer levels than possible using existing approaches;
(2) generation of levels for platformers like *Mega Man* that progress in multiple directions; and
(3) generation of blended levels combining platformers progressing in different directions.

FDG '20, September 15–18, 2020, Bugibba, Malta
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8807-8/20/09…$15.00
https://doi.org/10.1145/3402942.3409604

2 RELATED WORK

Methods for PCGML [30] attempt to generate new content by sampling models that have been trained on game data with the hope of producing content that is novel but also captures the patterns and properties of the games used for training. While numerous ML techniques (including autoencoders [12], LSTMs [28], Markov models [4, 25] and Bayes Nets [9, 27]) have been used for generating levels, more recent ML advances such as latent variable models like Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) are also being increasingly used for generating game content. Volz et al. [33] used a GAN to generate Mario levels and demonstrated the feasibility of using the learned latent space for evolving levels with desired properties. In a similar vein, Sarkar et al. [23] used VAEs to generate level segments that blended the properties of *Super Mario Bros.* and *Kid Icarus* and exhibited the utility of the VAE's latent space for evolving blended content with desired characteristics. In addition to Mario, GANs have also been used to generate *Doom* levels [6] and *Legend of Zelda* dungeons [8] while VAEs have been used to generate *Lode Runner* levels [32]. However, due to the nature of GANs and VAEs, all such approaches are required to work with fixed-size inputs and outputs and are thus forced in many cases to train on and generate segments of levels rather than levels themselves, since using entire levels as the fixed-size inputs would normally lead to an insufficient amount of training data. In GAN-based Mario generation [33], authors prescribe generating whole levels by simply stitching together generated segments. While this works due to the simple nature of Mario only progressing along one direction, it is not ideal since segments are generated from latent space vectors, each of which independently encodes a single segment, and thus there is no guarantee that successively generated segments follow each other optimally. Additionally, such an approach would not work for more complex platformers which can progress in multiple directions or to blend together segments from different games as in prior work [23] which ignores generating whole levels and restricts generation to segments alone. Thus, in this paper, we attempt to address the problem of generating continuous, whole levels for a variety of platformers while contending with the fixed-sized limitations of latent variable models. Recently, Gutierrez et al. [8] also addressed this issue of generating entire levels using fixed-sized inputs and outputs. In their work, they use GANs to generate fixed-size rooms and connect them using a graph grammar to form a dungeon. Our approach differs in that it uses VAEs, works with platformers and conditions the generation of a segment on the previous segment.

Another recent trend in PCGML research has been to focus on more creative uses of ML [11] for generating game content. Such works try to move beyond generating levels for an existing game or domain in order to enable PCG techniques such as game blending, domain transfer and automated game generation. These methods fall under combinational creativity [1], the branch of creativity in which existing concepts and domains are combined to generate novel ones. Previous such methods include Snodgrass and Ontañón's domain transfer work [24], Guzdial and Riedl's conceptual expansion technique [10] that generates new games by combining levels and rules of existing games using Bayes nets and game graphs, Snodgrass and Sarkar's [26] hybrid model combining binary space partitioning and VAEs to generate blended levels across multiple platformer games using a sketch representation and Sarkar et al.'s [23] use of VAEs for blending level segments of *Super Mario Bros.* and *Kid Icarus.* Our work builds directly on the latter by enabling generation of entire blended levels rather than just segments and additionally generating blended levels that are mostly traversable unlike the blended segments in the previous work.

3 METHOD

We describe the game level data used in this work as well as the two main parts to the generative pipeline—the VAE for segment generation and the random forest classifier for segment placement classification.

3.1 Level Data

We demonstrate our approach using levels from 3 classic NES platformers—*Super Mario Bros.* [18], *Kid Icarus* [19] and *Mega Man* [3], henceforth referred to as SMB, KI and MM respectively. Additionally, to test blending, we also used a combined SMB-KI domain. All levels were taken from the Video Game Level Corpus (VGLC) [31]. These levels use a tile-based text representation and are annotated with the path of an A^* agent tuned using the jump arcs of the game [29]. This allows a trained model to produce such paths in the generated levels and thus help make them playable. While SMB and KI levels progress exclusively left-to-right and bottom-to-top respectively, MM levels progress left-to-right as well as both bottom-to-top and top-to-bottom. Thus, past approaches for VAE and GAN-based level generation would not be able to reliably generate MM levels since consecutive randomly generated segments could be oriented incompatibly, which we address by conditioning on the previous segment in our approach. Similarly, levels in a blended SMB-KI domain would be expected to progress both to the right and to the top and would necessitate a similar modification in order to be amenable for generation. For all games, we train our models on 16x16 segments produced by sliding a window of that size horizontally and vertically across levels as appropriate given the orientation of the game. Horizontal segments of SMB and MM are originally 14 and 15 rows high respectively so we pad them with additional row(s) of all background tiles to have a uniform height of 16. This gave us 2458 segments for SMB, 1046 segments for KI and 1572 segments for MM. For the blended SMB-KI domain, to better balance the number of segments, we doubled the KI segments to end up with 2092. Additionally, for levels in the blended domain, we used the original tiles from each game's VGLC representation except for using a common tile for background and path. For all domains, paths are represented using a Mario character sprite.

3.2 Sequential Segment Generation using VAEs

To build our models, we trained a VAE on each of SMB, KI and MM as well as on the blended SMB-KI domain. VAEs [13] learn continuous, latent representations of data, and consist of an encoder network which learns to map data to a lower-dimensional vector in the latent space and a decoder network which learns to reconstruct the original data from this latent vector. This is achieved by training via minimizing a loss function that consists of two terms: 1) the reconstruction error and 2) the Kullback-Leibler (KL) divergence—a

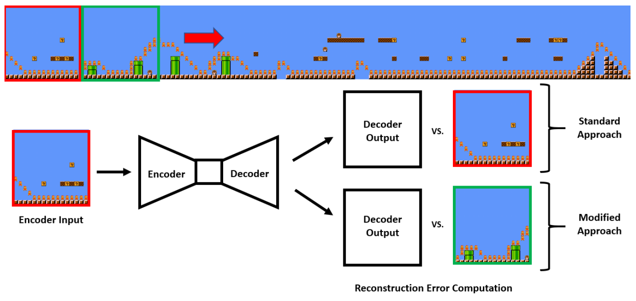


Figure 1: Comparison of reconstruction error computation between the standard approach and our modified approach.

statistical measure of the similarity between two probability distributions. Minimizing the former reduces the error between inputs and reconstructed outputs produced by the decoder while minimizing the KL divergence between the latent distribution and a known prior (typically a Gaussian) enforces the latent space to model a continuous, informative distribution. We modify the computing of the reconstruction error to enable our approach.

Our input consists of 16x16 segments. Typically, the reconstruction error would be computed between the segment output by the decoder and the corresponding segment that was passed through the encoder. Instead, in our approach, we compute this error between the decoder output and the segment that *follows* the corresponding segment that was encoded. This simple modification enables the VAE to learn a sequential model of segment generation where the encoder maps a given segment into a latent vector but the decoder maps that latent vector into the segment that would sequentially follow the original segment in a level. This is depicted in Figure 1. The algorithm shown below thus enables generation of a sequence of segments that follow a logical gameplay progression and hence can be combined into a coherent level.

Algorithm 1 GenerateLevel(init_segment, n)

```
Initialize $level$ to $init\_segment$
$num\_segments = 1$
$segment = init\_segment$
while $num\_segments \leq n$ do
    $z \leftarrow Encoder(segment)$
    $segment \leftarrow Decoder(z)$
    Add $segment$ to $level$
    $num\_segments$ += 1
end while
return $level$
```

All models were trained using PyTorch [20] and used the same architecture. Encoders and decoders each consisted of 4 linear layers with ReLU activation. The decoder output was further passed through a sigmoid layer. All models used a 128-dimensional latent space and were trained for 10000 epochs with the Adam optimizer and a learning rate of 0.001 decayed by 0.1 every 2500 epochs. Additionally, to aid in training, the weight of the KL-divergence term in the variational loss was annealed linearly from 0.0 to 1.0 over the first 2500 epochs.

3.3 Placement Classification

While the above modification gives us a sequential segment-based level generation model, it still only improves upon existing generation approaches along one direction. To generate levels that can dynamically progress along any direction, we need to determine where to place a generated segment in relation to the previous segment. For this purpose, for each domain, we train a random forest classifier on its segments. Here the inputs are the 16x16 segments and each label is the direction where the next segment appears in the original level. Thus, given a generated segment, the classifier determines in which direction (up, down, left or right) it should be placed with respect to the previous segment. We trained the classifiers using a 70%-30% train-test split for each domain, obtaining accuracies of 100% on each of SMB, KI and SMB-KI and 98.73% for MM. For the MM training set, we oversampled the vertically progressing segments till we had an equal number of segments in all directions to account for class imbalance. Combining the classifier with the VAE, gives us the following generative pipeline.

Algorithm 2 GenerateLevelWithDirs(init_segment, n)

```
$level \leftarrow$ GenerateLevel(init_segment, n)
$level\_with\_dirs \leftarrow \emptyset$
for $segment$ in $level$ do
    $dir \leftarrow Classifier(segment)$
    Add $(segment, dir)$ to $level\_with\_dirs$
end for
return $level\_with\_dirs$
```

Note that for the domains we used, only MM and the SMB-KI blend require a classifier. All levels in SMB and KI progress in one direction (right and up respectively) so in these cases, it is sufficient to simply place the generated segments one after another to get a coherent level. However for generality, we still used the classifier for every domain in all of our evaluations. A potential pitfall of using the classifier is dealing with segments that are incorrectly classified. The most common form of mis-classification that could disrupt the progression of a level is along an individual axis (i.e. segments that should be followed in the downward direction misclassified as upward and vice-versa; segments that should be followed to the right being misclassified as left and vice-versa). These are common because often times such segments taken individually could make sense in either direction. Segments that proceed upward and downward in MM for example, often share similar structures. Similarly, an individual segment in SMB taken in a vacuum could equally progress to the left or to the right. In such cases, it is the progression of the level generated up to that segment that determines the correct direction for the next segment rather than the segment itself. To account for this potential issue, we prevent the classifier from predicting the next direction to be the direction that connects the newly generated segment to the previous one i.e. the direction that if the newly generated segment were to be placed using it, would overwrite the previous segment. This is done by using the classifier prediction with the second highest likelihood if the prediction with the highest is the direction to avoid.

4 RESULTS

We tested our approach using a three-part evaluation, focusing on 1) the continuous nature of generated levels compared to past methods, 2) properties of generated blended levels and 3) the quality of generated levels that are arbitrarily long. We describe each part in the following sections.

4.1 Discontinuity

To test if levels generated using our methods have a better sense of progression than past methods, we introduced a *Discontinuity* metric. We define this as the absolute distance between path tiles along the adjoining edge of two successive segments. That is, if two segments are connected horizontally, we compute the displacement between the path tiles on the columns at the edge connecting the two segments. If either column does not have a path tile, the metric returns 16 by default since the maximum height of a column is 16 and thus the path tiles can be at most 15 tiles from each other. For segments connected vertically, this is similarly computed using the rows at the edge of the two segments. Thus, lower the value, the more continuous the path is from one segment to the next. The reasoning for this metric is that levels with a better sense of progression would have a more continuous path through its segments rather than a path with a lot of displacement between where it ends for one segment and where it begins for the next. While by no means a perfect measure of progression through a level, it nevertheless gives a sense of the nature of the path through a level and is thus suitable for comparison with past generative methods. For our computations, we ignored the fact that KI levels wrap around horizontally.

For evaluation, we generated 100 levels each for SMB, KI, MM and SMB-KI using two methods: 1) using the generative loop described in Algorithm 2, where segments are generated conditioned on the previous, and 2) stitching together segments generated independently of each other from randomly sampled latent vectors, which is how whole levels are generated using past methods. For the rest of the paper, we refer to these methods as *sequential* and *independent* respectively. Each generated level consisted of 12 segments for SMB and KI and 16 for MM since those were respectively the average number of 16x16 segments in levels from the original games. For combined SMB-KI, generated levels consisted of 12 segments. Results are shown in Table 1.

For all games, the sequential method led to significantly lower *Discontinuity* values than the independent method, thus suggesting that levels generated using the former have more continuous paths through their segments. Note that the differences are greater for MM and SMB-KI since as we have discussed, segments generated using the independent method for these two often leads to levels that are not traversable where as for SMB and KI, the independently generated levels are often playable even if not as continuous as sequentially generated ones.

Examples of sequential and independently generated levels for SMB are in Figures 4 and 5, for KI in Figure 6, for MM in Figures 8 and 9, and for SMB-KI in Figure 7. Based on visual inspection, the sequential method generates levels with a more continuous flow from segment to segment than the independent method. While expected, it is worth noting that using the sequential method, starting generation with the initial segment of an original level produces a level very similar to the original, as seen in Figures 5, 6(b) and 9, while the independent method of course does not do so since segments are generated independently. A byproduct of independent segment generation is that independently generated levels seem to be more diverse, less predictable and result in more directional changes for games that progress in multiple directions. In the right context, these are desirable features and thus it would be interesting in the future to look at methods capable of trading off between the better playability and continuity of the sequential method with the increased variety of the independent approach.

Game	Sequential	Independent
SMB	3.86 ± 2.28	5.91 ± 2.04
KI	3.99 ± 2.59	7.37 ± 1.99
MM	6.54 ± 2.63	11.18 ± 1.69
SMB-KI	5.4 ± 2.42	9.84 ± 1.76

Table 1: Average per-segment *Discontinuity* values along with standard deviation. A Wilcoxon Rank Sum Test showed differences to be significant with $p < .001$ in all cases.

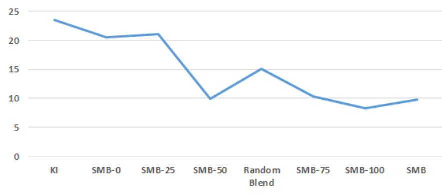
4.2 Blending

The ability to generate levels progressing in multiple directions also enables us to move beyond generating segments that blend games progressing in different directions to entire levels that do so. To test blending, we generated 100 12-segment blended SMB-KI levels using a VAE trained on segments from both games as described previously. We generated 6 sets of 100 such levels with each set differing in the choice of initial segment. The first set used initial segments sampled randomly from the SMB-KI latent space. The remaining five used initial segments obtained by interpolating between latent vectors corresponding to actual SMB and KI segments at intervals of 25%. Thus we obtained 5 such sets labeled as SMB-0, SMB-25, SMB-50, SMB-75 and SMB-100, indicating the distance interpolated from the KI segment to the SMB segment. We compared the blended levels with the original SMB and KI levels using the following tile-based segment-level metrics:

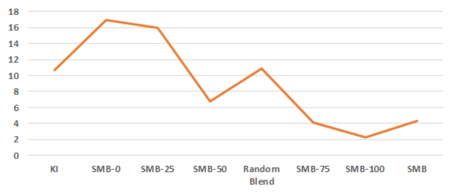
- *Density*: the proportion of a segment occupied by tiles that the player can stand on such as blocks, ground and platforms
- *Non-Linearity*: a measure of how a segment's topology fits to a line, calculated by computing the mean square error of running linear regression on the topmost point of columns in a segment. Zero value indicates perfect linearity

Blend	SMB	KI
SMB-0	0.5	99.5
SMB-25	4	96
SMB-50	86.1	13.9
SMB-75	85	15
SMB-100	94.3	5.7
Random Blend	43.4	56.6

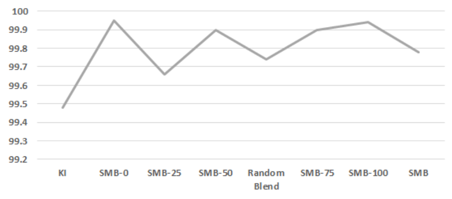
Table 2: Percentage of segments (out of $100x12 = 1200$) classified as SMB-like and KI-like using the directional classifier.



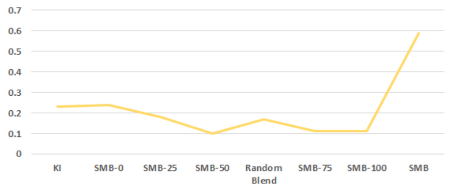
(a) Density



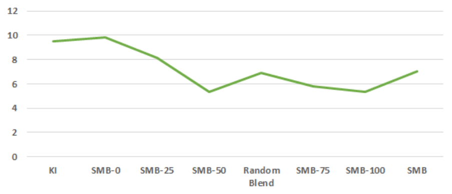
(b) Non-Linearity



(c) Leniency

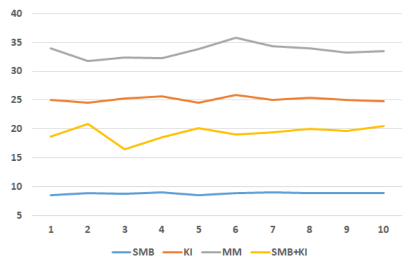


(d) Interestingness

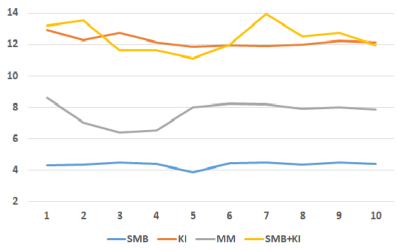


(e) Path-Prop

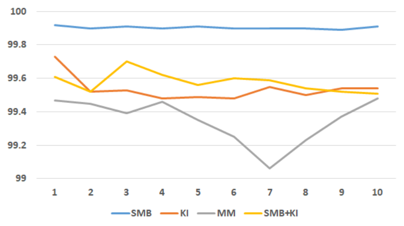
Figure 2: Per-segment tile metrics for original SMB and KI levels along with different types of blends.



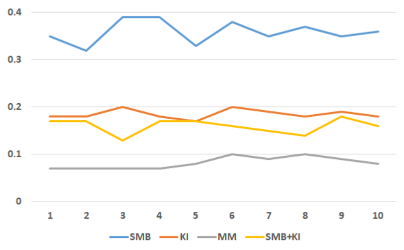
(a) Density



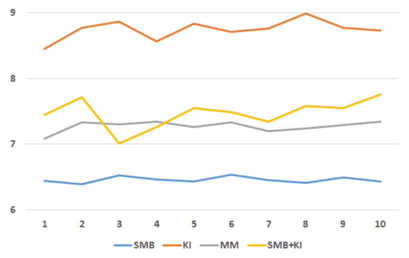
(b) Non-Linearity



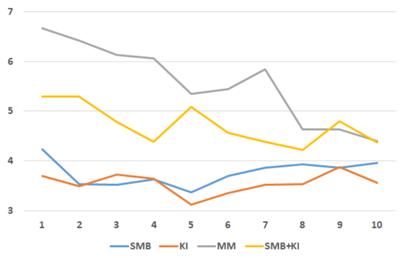
(c) Leniency



(d) Interestingness



(e) Path-Prop



(f) Discontinuity

Figure 3: Per-segment metric values plotted for each grouping of 16 segments for MM and each grouping of 12 segments for the other games. x-axis values indicate 1st such grouping, 2nd such grouping etc. y-axis indicates average metric value for the corresponding group of segments.

- *Leniency*: the proportion of a segment that is not occupied by any enemy or hazard tiles
- *Interestingness*: the proportion of a segment occupied by interactable items such as collectables and powerups
- *Path-Prop*: the proportion of a segment occupied by path tiles

Results are given in Figure 2 and show that the values for blended levels fall mostly between those for SMB and KI, suggesting that the properties of the generated levels do blend those of the two original games, especially in terms of *Density*, *Path-Prop* and *Nonlinearity*. Values for *Leniency* and *Interestingness* for blended levels are between those for SMB and KI as well but not in the expected pattern and speaks to generated levels having fewer enemies, hazards and collectible items than the originals.

Additionally, we also looked at the proportion of SMB-like and KI-like segments that were generated for the different sets of blends. Since we know that SMB and KI levels progress exclusively to the right and upward respectively, we can use our directional classifier as a proxy for this purpose. That is, blended segments classified to have the next segment to the right can be deemed to be more

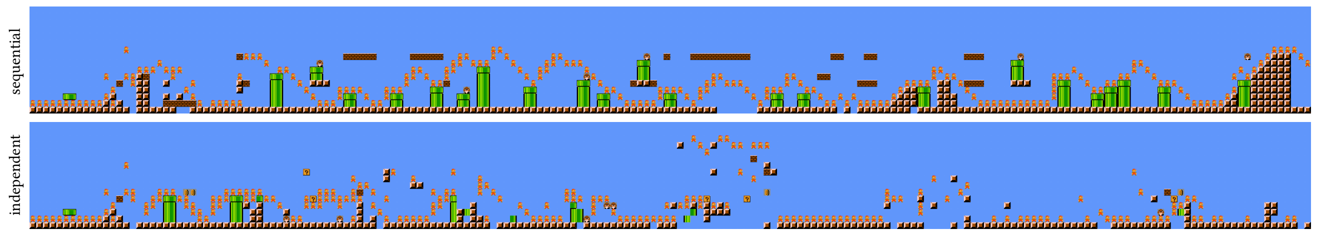


Figure 4: Example SMB levels generated using sequential (above) and independent (below) methods starting with the same randomly generated initial segment.

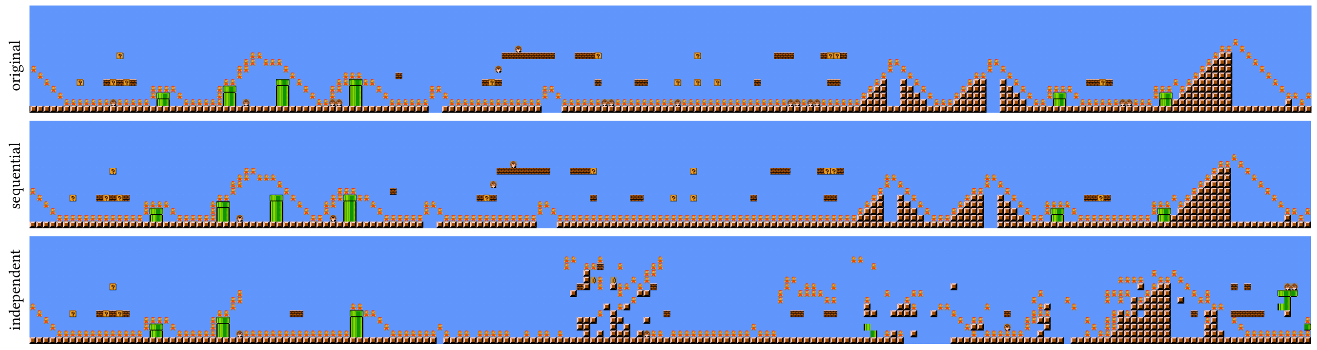


Figure 5: Original SMB level from the VGLC [31] (top) and example levels generated with the initial segment of the original using the sequential (middle) and independent (below) methods.

SMB-like while those classified to have the next segment be to the top can be deemed KI-like. Results for this are shown in Table 2. We see that levels generated from random latent vectors perform the most amount of blending while those generated from interpolated vectors heavily favor the game with the higher proportion in the interpolation, mostly ignoring the actual proportion itself. Thus, while this approach is successful in generating blended levels, in the future, we would like to augment it such that the nature and amount of blending is more controllable.

4.3 Progression

For our final evaluation, we wanted to demonstrate the ability of our approach to generate arbitrarily long levels without the quality or characteristics of later segments deteriorating. We generated 100 levels of 120 segments for each of SMB, KI and SMB-KI and 100 levels of 160 segments for MM (i.e. levels approximately 10 times the size of an average level) and computed the average *Discontinuity* and tile-based metrics defined above per segment for each of the 10 sub-levels (i.e. each set of 16 segments in MM and each set of 12 segments in the others). Results for this are shown in Figure 3. Ideally, we would like to see little variation in terms of metric values for all 10 sets of segments and this indeed bears out in the results for all metrics and games except for *Discontinuity* for MM which actually seems to get lower as more segments are generated. A possibility for this is that the MM model might be falling into a pattern of generating similar (or the same) segments over and over again causing little variation in path and hence low *Discontinuity*. This ties into the broader problem of VAEs suffering from *posterior collapse* [14, 15, 21] where the decoder learns to reconstruct data while ignoring a subset (or in the worst case, all) of the latent dimensions, resulting in an uninformative latent space. Though we trained our models using KL-annealing [2, 5] to specifically account for this, and we do not encounter it for regular-sized levels, it is possible that the problem manifests itself when trying to generate larger levels and needs to be studied more thoroughly in the future.

5 CONCLUSION AND FUTURE WORK

In this work, we presented a novel PCGML approach that combines the use of a variational autoencoder and a random forest classifier to produce a model for sequential platformer level generation. Our results demonstrate that this enables generation of more coherent platformer levels than past approaches, generation of platformer levels that progress in multiple directions, blending of levels from games that progress differently and generation of levels that are arbitrarily long without suffering from a loss of quality. There are several considerations for future work.

While we evaluated our model in terms of continuity of generation and blending, we did not specifically evaluate the classifier on generated segments, mainly in part due to generated segments not having a ground truth to test against. In the future, we could compare classifier-based segment placement with other placement strategies, optimizing for metrics such as continuity and playability.

Our approach demonstrated the feasibility of generating levels for multi-directional platformers such as *Mega Man* but there is massive room for improvement in terms of quality of models and generated levels, reliability of generation and controllability, all of

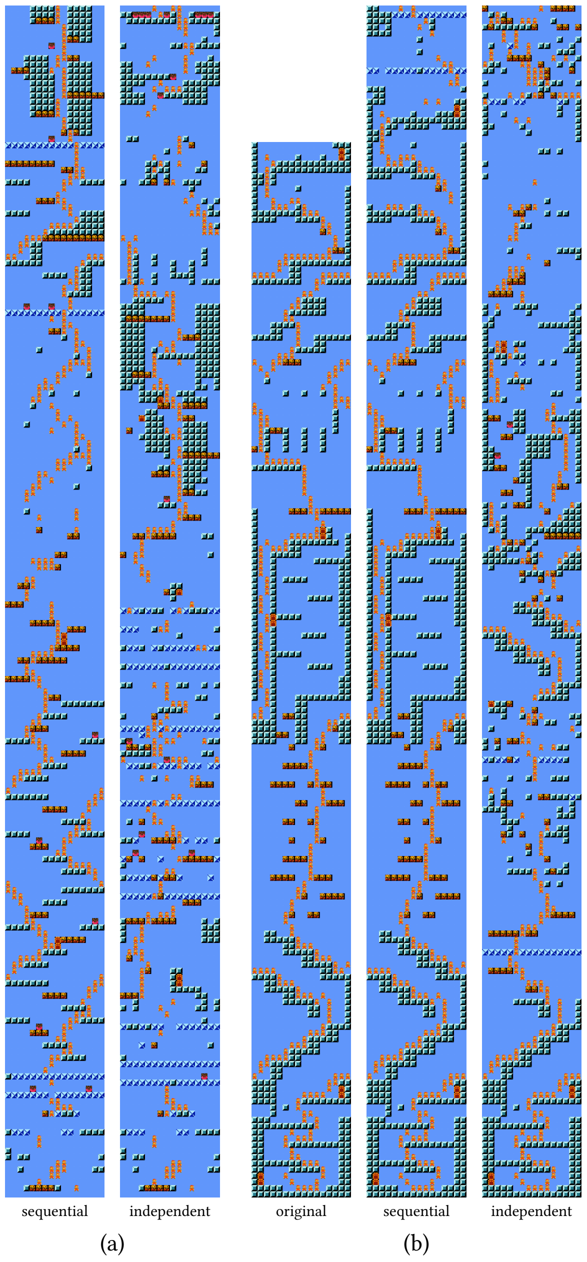


Figure 6: (a) Example KI levels generated using sequential (left) and independent (right) methods starting with same randomly generated initial segment. (b) Original KI level from the VGLC [31] (left) and example levels generated with initial segment of original using the sequential (middle) and independent (right) methods.

which should be investigated in future work. Moreover, none of the games we used had levels progressing from right-to-left. While our classifier was trained to work with all four directions in mind and should work as is for right-to-left progression, this needs to be empirically validated in the future.

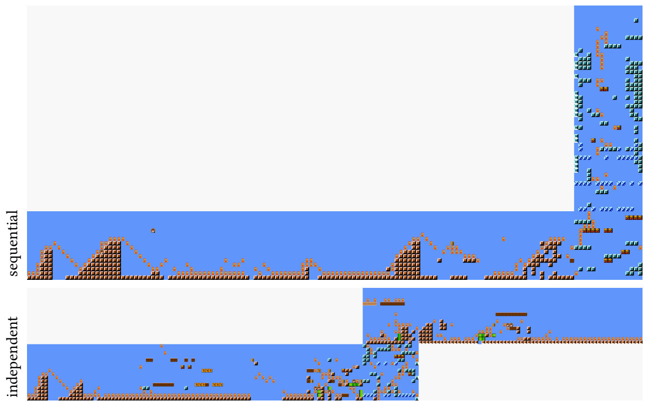


Figure 7: Example blended SMB-KI levels generated using the sequential (above) and independent (below) methods starting with the same randomly generated initial segment.

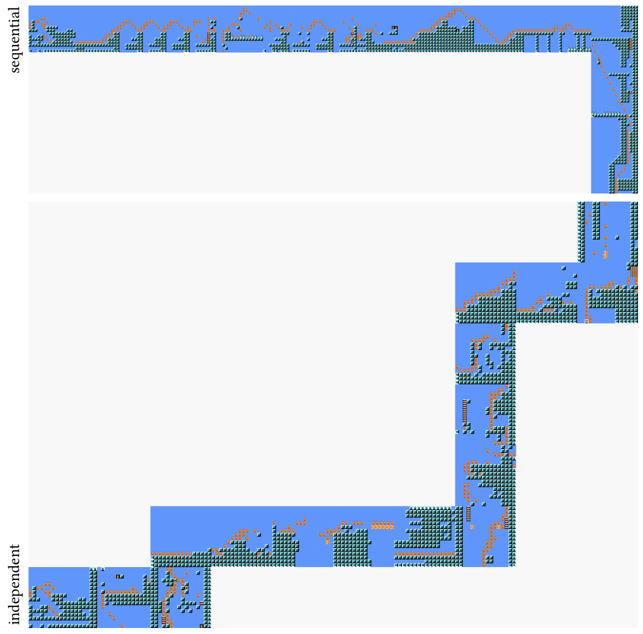


Figure 8: Example MM levels generated using the sequential (top) and independent (bottom) methods with the same randomly generated initial segment.

Additionally, while our method demonstrably allows for generation of traversable, blended levels, the current approach does not have any direct means of controlling the blend proportions. It is possible to use some of the related latent variable evolution strategies from [23] but this needs to be empirically tested in future work. To this end, we could use conditional variants of the VAE [17]. Such models can explicitly condition the generation of segments on properties such as direction and game type and thereby allow designers to generate segments with greater control.

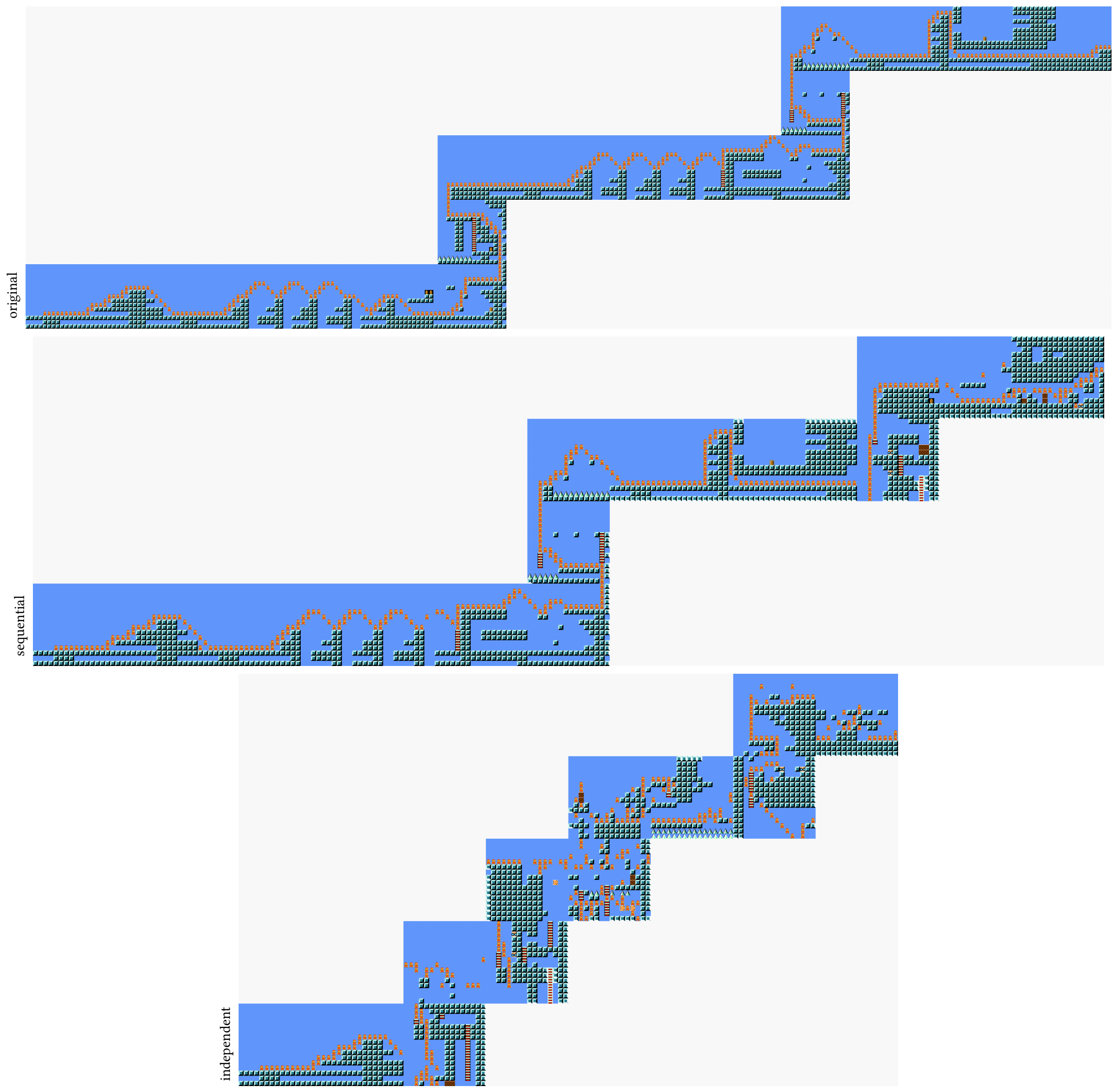


Figure 9: Original MM level from the VGLC [31] (top) and example levels generated with the initial segment of the original using the sequential (middle) and independent (bottom) methods.

Finally, we intend to test this approach with more games and more diverse blends composed of more than two games. The ability to generate traversable blended levels opens up the possibility of generating new mechanics which combined with these newly blended levels could form the foundation for generating entire blended games in the future.

REFERENCES

[1] Margaret A. Boden. 2004. *The Creative Mind: Myths and Mechanisms*. Psychology Press.

[2] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349v4* (2016).

[3] Capcom. 1987. *Mega Man*. Game [NES].

[4] Steve Dahlskog, Julian Togelius, and Mark J Nelson. 2014. Linear levels through n-grams. *Proceedings of the 18th International Academic MindTrek* (2014).

[5] Hao Fu, Li Chunyuan, Liu Xiaodong, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. 2019. Cyclical annealing schedule: a simple approach to mitigating KL vanishing. *arXiv preprint arXiv:1903.10145* (2019).

[6] Edoardo Giacomello, Pier Luca Lanzi, and Daniele Loiacono. 2018. Doom Level Generation using Generative Adversarial Networks. In *IEEE Games, Entertainment, Media Conference (GEM)*.

[7] Ian Goodfellow, Jean Abadie-Pouget, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*.

[8] Jake Gutierrez and Jacob Schrum. 2020. Generative Adversarial Network rooms in generative graph grammar dungeons for The Legend of Zelda. *arXiv preprint arXiv:2001.05065v1* (2020).

[9] Matthew Guzdial and Mark Riedl. 2016. Learning to blend computer game levels. *arXiv preprint arXiv:1603.02738* (2016).

[10] Matthew Guzdial and Mark Riedl. 2018. Automated game design via conceptual expansion. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[11] Matthew Guzdial and Mark Riedl. 2018. Combinatorial creativity for procedural content generation via machine learning. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[12] Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. 2016. Autoencoders for level generation, repair and recognition. In *Proceedings of the ICCC Workshop on Computational Creativity and Games*.

[13] D.P. Kingma and M. Welling. 2013. Auto-encoding Variational Bayes. In *The 2nd International Conference on Learning Representations (ICLR)*.

[14] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. 2019. Don't blame the ELBO! A linear VAE perspective on posterior collapse. In *33rd Conference on Neural Information Processing Systems (NeurIPS)*.

[15] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. 2019. Understanding posterior collapse in generative latent variable models. In *International Conference on Learning Representations (ICLR)*.

[16] Simon M. Lucas and Vanessa Volz. 2019. Tile pattern KL-divergence for analysing and evolving game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 170–178.

[17] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Networks. *arXiv preprint arXiv:1411.1784* (2014).

[18] Nintendo. 1985. *Super Mario Bros.* Game [NES].

[19] Nintendo. 1986. *Kid Icarus*. Game [NES].

[20] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

[21] Ali Razavi, Aaron van den Oord, Ben Poole, and Oriol Vinyals. 2019. Preventing posterior collapse with Delta-VAEs. In *International Conference on Learning Representations (ICLR)*.

[22] Anurag Sarkar and Seth Cooper. 2018. Blending levels from different games using LSTMs. In *2018 Experimental AI in Games Workshop*.

[23] Anurag Sarkar, Zhihan Yang, and Seth Cooper. 2019. Controllable level blending between games using Variational Autoencoders. In *2019 Experimental AI in Games Workshop*.

[24] Sam Snodgrass and Santiago Ontañón. 2016. An approach to domain transfer in procedural content generation of two-dimensional videogame levels. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[25] Sam Snodgrass and Santiago Ontañón. 2017. Learning to generate video game maps using Markov Models. *IEEE Transactions on Computational Intelligence and AI in Games* (2017).

[26] Sam Snodgrass and Anurag Sarkar. 2020. Multi-domain level generation and blending with sketches via example-driven BSP and Variational Autoencoders. In *Proceedings of the 15th Conference on the Foundations of Digital Games*.

[27] Adam Summerville, Matthew Guzdial, Michael Mateas, and Mark O Riedl. 2016. Learning player tailored content from observation: platformer level generation from video traces using LSTMs. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[28] Adam Summerville and Michael Mateas. 2016. Super Mario as a String: Platformer Level Generation Via LSTMs. *Proceedings of 1st International Joint Conference of DiGRA and FDG* (2016).

[29] Adam Summerville, Joe Osborn, Christoffer Holmgard, and Daniel W. Zhang. 2017. Mechanics automatically recognized via interactive observation: jumping. In *Proceedings of the Twelfth International Conference on Foundations of Digital Games*.

[30] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* (2018).

[31] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. 2016. The VGLC: The video game Level Corpus. In *Seventh Workshop on Procedural Content Generation at First Joint International Conference of DiGRA and FDG*.

[32] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and evolutionary algorithm for level generation in Lode Runner. In *IEEE Conference on Games*.

[33] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving Mario levels in the latent space of a deep convolutional Generative Adversarial Network. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 221–228.