Conditional Level Generation and Game Blending

Anurag Sarkar[1], Zhihan Yang[2] and Seth Cooper[1]
[1]Northeastern University
[2]Carleton College
sarkar.an@northeastern.edu, yangz2@carleton.edu, se.cooper@northeastern.edu

Abstract

Prior research has shown variational autoencoders (VAEs) to be useful for generating and blending game levels by learning latent representations of existing level data. We build on such models by exploring the level design affordances and applications enabled by conditional VAEs (CVAEs). CVAEs augment VAEs by allowing them to be trained using labeled data, thus enabling outputs to be generated conditioned on some input. We studied how increased control in the level generation process and the ability to produce desired outputs via training on labeled game level data could build on prior PCGML methods. Through our results of training CVAEs on levels from *Super Mario Bros.*, *Kid Icarus* and *Mega Man*, we show that such models can assist in level design by generating levels with desired level elements and patterns as well as producing blended levels with desired combinations of games.

Introduction

Procedural Content Generation via Machine Learning (PCGML) (Summerville et al. 2018) has emerged as a viable means of building generative models for game levels by training on levels from existing games. While several ML approaches have been utilized for PCG such as LSTMs (Summerville and Mateas 2016), Bayes nets (Guzdial and Riedl 2016a) and Markov models (Snodgrass and Ontañón 2017), a recent body of work has emerged that focuses on using latent variable models such as Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) and Variational Autoencoders (VAEs) (Kingma and Welling 2013). These models learn latent encodings of the input game levels, comprising a continuous latent space which can then be sampled and explored to generate new levels. Such models have been used both for level generation (Volz et al. 2018; Gutierrez and Schrum 2020) and level blending (Sarkar, Yang, and Cooper 2019; Snodgrass and Sarkar 2020). Additionally, attempts have been made to make such generation and blending controllable via latent vector evolution (Bontrager et al. 2018). This involves optimizing some objective function via evolutionary search in the learned latent space of the model to find vectors corresponding to levels with desired properties. Thus, controllability is achieved via evolutionary search once training has already been performed and is independent of the model. However, conditional variants of both GANs (Mirza and Osindero 2014) and VAEs (Sohn, Lee, and Yan 2015) could enable such controllability as part of the model itself. These variants allow models to be trained on labeled data and thereby allow generation to be conditioned on input labels. Thus, when applied for PCGML, such models could use labels provided by designers to produce controllable level generators without having to define objective functions and run evolution.

SMB - ⟨10011⟩ KI - ⟨1101⟩ MM - ⟨10101⟩

Figure 1: Example original segments with corresponding element labels. *Super Mario Bros* (SMB)-⟨*Enemy, Pipe, Coin, Breakable, ?-Mark*⟩, *Kid Icarus* (KI)-⟨*Hazard, Door, Moving Platform, Fixed Platform*⟩, *Mega Man* (MM)-⟨*Hazards, Door, Ladder, Platform, Collectable*⟩. 0/1 in labels indicate absence/presence of corresponding elements in the segment.

Thus, we train conditional VAEs (CVAEs) on levels from *Super Mario Bros.*, *Kid Icarus* and *Mega Man* using different sets of labels corresponding to the presence of various game elements as well as design patterns. Additionally, we train a combined model with levels labeled with the game they belong to. Our results show that CVAEs can generate levels with and without desired elements and design patterns, making it a promising model to inform future level design tools. Further, our results suggest that CVAEs can help perform controllable game blending using different labels provided during generation.

Background

Controllability has been the focus of much PCG research with several works developing generators that produce content based on designer preferences such as (Liapis, Yannakakis, and Togelius 2013; Alvarez et al. 2020; Smith

et al. 2009). Besides these systems, other works on controllable PCG have included graph-based methods (Valls-Vargas, Zhu, and Ontañón 2017; Dormans 2010) and evolutionary computation (Togelius et al. 2010). Similarly, a number of PCGML works have also incorporated controllability. Snodgrass and Ontañón (2016) used a constraint-based method to control sampling for a multi-dimensional Markov model of Mario levels while Sarkar and Cooper (2018) used multiple LSTM models with turn-based weighting to control generation of blended Mario-Kid Icarus levels. In terms of latent variable models, Volz et al. (2018) used CMA-ES to evolve vectors in the latent space of a GAN trained on Mario levels. This produced levels capturing desired characteristics based on the objective used for evolution. A similar method was used by Sarkar, Yang, and Cooper (2019) to generate and blend levels of Mario and Kid Icarus using a VAE trained on both games. Recently, Schrum et al. (2020) produced a tool enabling user-controlled generation of Mario levels and Zelda-like dungeons by evolving vectors via interactive exploration of the GAN latent space. In terms of conditional models, Torrado et al. (2019) combined a conditional GAN approach with self-attention mechanisms for controllable generation of GVGAI levels. Our work differs in using CVAEs instead of CGANs and not restricting to GVGAI. Moreover, their conditioning features were learned from input levels and not supplied externally as in our case, which seems more suited to future co-creative applications.

Prior work has demonstrated the utility of conceptual blending (Fauconnier and Turner 1998) for generating new levels and even entire games. Gow and Corneli (2015) presented a VGDL-based manual game blending framework to create new games by combining elements of existing ones. Guzdial and Riedl (2016b) similarly blended Mario levels to produce new levels. Sarkar and Cooper (2018) and Sarkar, Yang, and Cooper (2019) used LSTMs and VAEs respectively to blend levels of Mario and Kid Icarus while Sarkar et al. (2020) extended the latter to a larger set of games and incorporated playability into blended levels. In this work, we show that game blending can also be achieved using CVAEs.

Conditional VAEs (CVAE) (Sohn, Lee, and Yan 2015; Yan et al. 2015) allow VAEs (Kingma and Welling 2013) to be conditioned on attributes. Unlike VAEs which are trained in an unsupervised manner, to train CVAEs, each input datapoint is associated with a label. The encoder learns to use this label to encode the input into the latent space while the decoder learns to use the label to decode the latent encoding. Thus, such models when trained on game levels could allow generation to be conditioned on designer-specified labels. Further, the same latent vector can produce different outputs by varying the conditioning labels. This enables additional affordances for level design and allows generation to be controlled without having to perform evolutionary search.

Method

Level Data and Conditioning

We used level data from the Video Game Level Corpus (VGLC) (Summerville et al. 2016) for the classic NES platformers *Super Mario Bros.* (SMB), *Kid Icarus* (KI) and *Mega Man* (MM). For training our models, we used 16x16 level segments cropped out from the levels of each game by using a sliding window. To enable this, SMB levels and horizontal portions of MM levels were padded with 1 and 2 empty rows respectively. We finally obtained 2643 segments for SMB, 1142 for KI and 2983 for MM. Using these games, we looked at three different conditioning approaches based on 1) game elements 2) SMB design patterns and 3) game blending, motivated by wanting to generate levels containing desired elements, exhibiting desired patterns and consisting of desired combinations of games, respectively. Conditioning is accomplished by associating each input level segment with a corresponding label during the training process. In all cases, labels take the form of binary-encoded vectors.

Game Elements For game elements, we used a unique set of conditioning labels for each game. The label vector length was determined by the number of different game elements considered for each game with a 0/1 value for a vector element indicating the absence/presence of the corresponding game element in the corresponding level segment. For SMB, we considered *Enemy*, *Pipe*, *Coin*, *Breakable* and *Question Mark* elements, for KI, *Hazard*, *Door*, *Moving* and *Stationary Platforms* and for MM, *Hazard*, *Door*, *Ladder*, *Platforms* and *Collectables*. Thus, we used 5-element binary labels for SMB and MM and 4-element labels for KI, yielding $2^5 = 32$ unique labels for SMB and MM and $2^4 = 16$ unique labels for KI. Example segments and their corresponding element labels are shown in Figure 1.

Design Patterns For SMB design patterns, we picked 10 such patterns based on the 23 described by Dahlskog and Togelius (2012). The ones we consider are:

- *Enemy Horde (EH):* group of 2 or more enemies
- *Gap (G):* 1 or more gaps in the ground
- *Pipe Valley (PV):* valley created by 2 pipes
- *Gap Valley (GV):* valley containing a *Gap*
- *Null (empty) Valley (NV):* valley with no enemies
- *Enemy Valley (EV):* valley with 1 or more enemies
- *Multi-Path (MP):* segment split into multiple parts horizontally by floating platforms
- *Risk-Reward (RR):* segment containing a collectable guarded by an enemy
- *Stair Up (SU):* ascending stair case pattern
- *Stair Down (SD):* descending stair case pattern

We thus had 10-element binary labels for a total of $2^{10} = 1024$ possible unique labels though a vast majority of these do not occur in the data. For this, we only trained on SMB levels since we only used SMB design patterns.

Game Blending For game blending, we trained on segments from all 3 games taken together with labels indicating which game the segments belonged to. We used a 3-element label with the 1st, 2nd and 3rd elements indicating if the segment was from SMB, KI or MM respectively.

Generation and Blending using CVAE

Training CVAEs involves associating each input data instance (in our case, level segments) with a label vector. An input instance is concatenated with its label and passed

through the encoder to obtain a latent vector. The same label is then concatenated with this latent vector and passed through the decoder. Thus, the encoder learns to use the provided labels to learn latent encodings of the data and similarly, the decoder learns to use the same labels to decode the encodings. Since the encoder and decoder use the labels to learn the encodings between the latent space and the data, the same latent vector could be made to produce different outputs by varying the label during generation. These affordances of the conditional VAE could inform level design and generation in two specific ways: 1) enable controllable generation by using labels to produce desired content and 2) generate variations of existing content by encoding it into latent space and decoding it using different labels. Through this work, we hoped to explore these specific affordances.

For game elements, we trained separate CVAEs for each game. In all models, both the encoder and decoder consisted of 4 fully-connected layers with ReLU activation. For SMB and MM, the conditioning labels were binary vectors of length 5 while for KI, they were of length 4. Labels for each input segment were determined by checking for the presence of the relevant game elements within that segment and assigning 0 or 1 to the corresponding element of the label. All models were trained in Pytorch (Paszke et al. 2017) for 10000 epochs using the Adam optimizer with a learning rate of 0.001 decayed by 0.1 every 2500 epochs.

For design patterns, we trained only on SMB data. Rather than use all segments obtained by sliding the window across the levels with redundancy in terms of overlap, we only used non-overlapping segments. We found non-overlapped segments better preserve the original design patterns. Since this led to fewer segments, we additionally used VGLC level data from *Super Mario Bros II: The Lost Levels*, to obtain a total of 407 segments. Labels corresponding to design patterns were assigned manually based on visual inspection. The model architecture was the same as those used for game elements, but was only trained for 5000 epochs with the learning rate decay occurring every 1250 epochs.

Finally for blending, we trained on levels from SMB, KI and MM taken together. Labels indicated the game that the levels belonged to and were of length 3 with $\langle 100 \rangle$, $\langle 010 \rangle$ and $\langle 001 \rangle$ indicating SMB, KI and MM respectively. Here, we intend to achieve blending by leveraging the fact that it is possible to condition generation using labels that do not appear in the training set. For instance, though the 3 labels above are the only ones used in the training set, we could still, for example, use label $\langle 110 \rangle$ for conditioning and expect to generate a level that blends SMB and KI. The model architecture was similar to that used for game elements. We duplicated the number of training segments for KI to better match the number of segments for SMB and MM.

In each of the above cases, we trained 3 versions of each model, consisting of latent spaces of size 32, 64 and 128.

Results

We performed a three-part evaluation focusing on each of the conditioning cases described above. Note that for KI and MM figures, we reuse certain sprites from SMB. Paths for all games are shown using a Mario character sprite.

	32-Dim		64-Dim		128-Dim	
	Exact	None	Exact	None	Exact	None
SMB-Rand	**33.5**	17.6	32.7	**17.1**	27.1	19
KI-Rand	**50.7**	**10**	42	10.5	41.4	9.8
MM-Rand	**17.4**	43.2	15.2	**42.5**	14.5	43.7
SMB-Train	**35.1**	**16.4**	33.6	**16.4**	28.3	17.5
KI-Train	**49.4**	**8.3**	40.3	9.3	39.5	8.5
MM-Train	**17.8**	42.7	15.3	**42.1**	14.8	42.6

Table 1: Results of conditioning randomly sampled ('Rand') and training ('Train') segments using element labels. Highest Exact values and lowest None values per game are highlighted in bold.

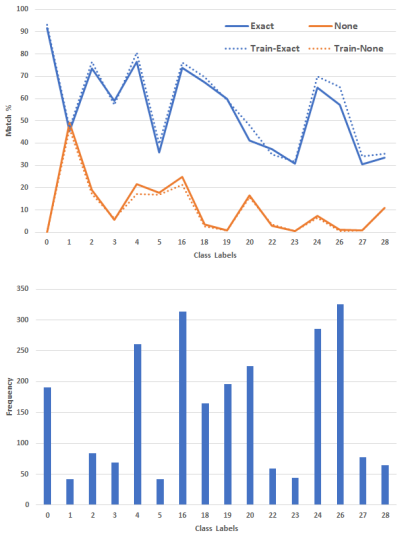

Figure 2: Results of game element conditioning for both generated and training levels of SMB (top) with frequencies of each label in the training data (bottom). X-axis values are the integer encodings of the equivalent binary label. Results shown for 16 most frequent labels in the training levels.

Game Elements

For evaluation, for each game, we randomly sampled 1000 latent vectors and conditioned the generation for each vector using each possible label (32 for SMB and MM, 16 for KI). We then tested if the generated segments contained the elements as prescribed by the conditioning labels i.e. compared the label for a generated segment with the conditioning label used to generate it. The label for the generated segment was determined using the same method for assigning labels to training segments. We computed the percentage of segments for which the output label was an exact match as the one used for conditioning, as well as the percentage where none of the elements that the label indicated should be present were actually present in the generated segment. We also performed this evaluation for the original level segments from each game by forwarding them through the encoder and decoder using each label. Percentages of exact and none matches averaged across all labels are given in Table 1.

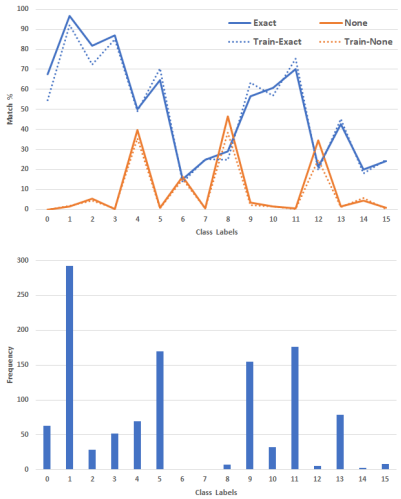

Figure 3: Results of game element conditioning for both generated and training levels of KI (top) with frequencies of each label in the training data (bottom). X-axis values are the integer encodings of the equivalent binary label.

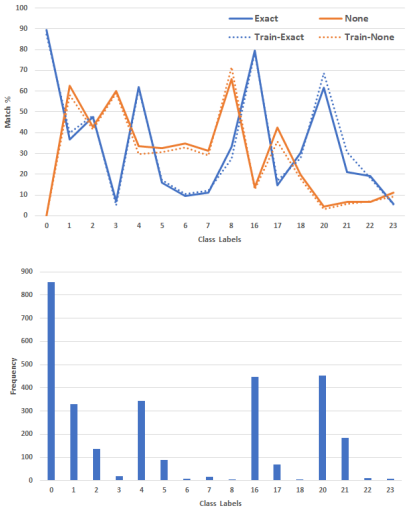

Figure 4: Results of game element conditioning for both generated and training levels of MM (top) with frequencies of each label in the training data (bottom). X-axis values are the integer encodings of the equivalent binary label. Results shown for 16 most frequent labels in the training levels.

For all games and for both random samples and training levels, the 32-dimensional model leads to the highest percentage of exact matches. The 64-dimensional models do better in producing lowest percentages of no matches but not too much better than 32. Interestingly, in most cases the 128-dimensional model exhibits the worst performance for these measures. Across games, results for KI are most promising in terms of both the highest percentage of exact matches and lowest percentage of cases with no matching elements while results for MM were the worst. The average Exact match percentages seem low primarily because the model produces few exact matches when conditioning with labels not in the training data. Thus we plotted match percentages for both generated and training segments obtained when using each label. Results for the 32-dimensional model for SMB, KI and MM are given in Figures 2, 3 and 4 respectively. Each plot also depicts the frequency of each label in the training data. Labels are shown along the horizontal axis using the integer representation for the equivalent binary encoding.

For all games, conditioning using labels that appear frequently in the training levels is a lot more reliable. Example levels generated using a selection of labels for conditioning using the 32-dimensional models are shown in Figures 5. These show that the affordances of the CVAE enable both controllable generation and the generation of novel variations of existing content. The top rows in each of these figures demonstrate how an existing segment could be edited into a new one by simply changing the conditioning label. This holds promise for co-creative applications as a list of designer preferences regarding the elements they want in generated segments could be converted to binary conditioning labels and then used to generate the desired segments.

Design Patterns

Evaluating design pattern conditioning was more challenging. Unlike for game elements, where input segments could be labeled automatically by checking for elements within the segment, labels for design patterns had to be assigned manually for each segment since we lack automated methods for identifying design patterns. Consequently, we could not automatically determine if the label for a generated segment matched with the label used to generate it. Moreover, due to the low number of training segments and high number of labels, training a classifier to determine a segment's design pattern was also not feasible. Thus, we restrict our evaluation in this case to visual inspection with Figure 6 showing example segments generated by conditioning on different labels corresponding to different combinations of design patterns. Results are shown for the 8 most common design pattern combinations in the training data. While not a robust evaluation, we see that the labels are reliable in producing the indicated design patterns in the generated segment.

Blending

To evaluate blending, we randomly sampled 1000 latent vectors and conditioned the generation of each using each of the 8 possible labels denoting the 8 possible combinations of the 3 games. We trained a random forest classifier on the training segments using the respective games as the class labels for the segment. We obtained a 99.12% classification accuracy using a 80%-20% train-test split. Note there are 2 label types: conditioning labels appended to latents to control generation, and game labels (SMB/KI/MM) predicted by the classifier indicating which game it thinks a generated segment belongs to. For each conditioning label, we then computed the percentage of generated segments that were

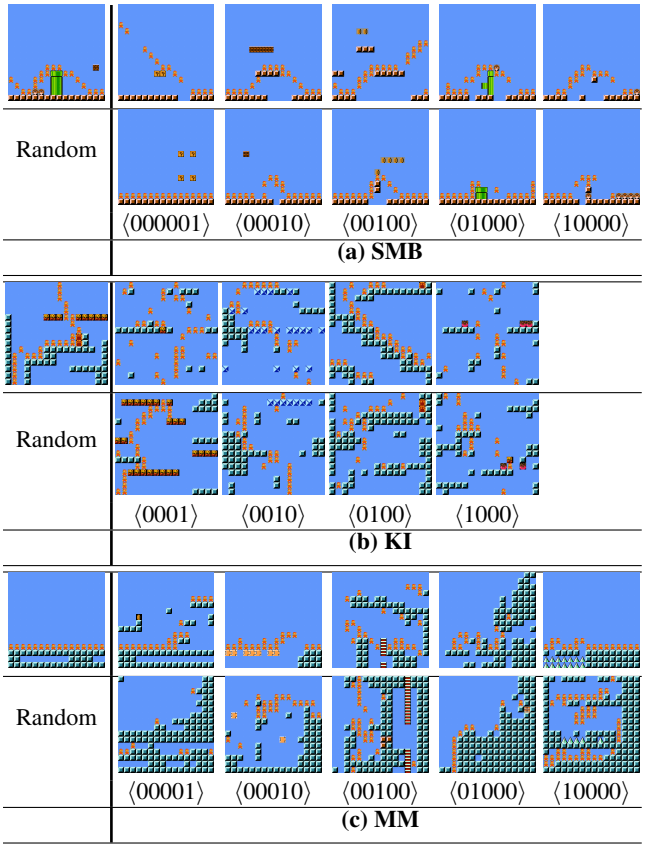

Figure 5: SMB, KI and MM segments generated by conditioning the original segment on the left (top) and a random vector (bottom) using the corresponding labels, as explained in Figure 1. Changing the label changes the content generated using the same the latent vector.

classified as SMB, KI or MM based on the classifier's predicted game label. When classifying segments generated via conditioning using one of the blended conditioning labels (i.e. SMB+KI, KI+MM, SMB+MM and SMB+KI+MM), we expect predicted game labels to be more spread across the 3 games versus when classifying segments conditioned using a single game conditioning label. For example, for segments generated using the SMB label (⟨100⟩), we expect a very high percentage to be classified as SMB and a very low percentage classified as others. For those generated using the blended labels, we would expect predictions with more variance. For example, for the SMB+MM label (⟨101⟩), we would expect most segments classified as SMB or MM (but not too many for either) and very few classified as KI. Results are given in Table 2 and are in accordance with expectations. For the original game labels (i.e.⟨100⟩, ⟨010⟩ and ⟨001⟩), a majority of segments (over 93% in all cases) are classified as the corresponding game. This drops and is more spread out as the blended labels (i.e. those with multiple 1s) are used. Also as expected, predictions are most evenly spread out for ⟨000⟩ and ⟨111⟩. Interestingly, in all cases other than ⟨000⟩, a game is predicted 10% or less if

	32-dim CVAE			64-dim CVAE			128-dim CVAE		
Label	SMB	KI	MM	SMB	KI	MM	SMB	KI	MM
⟨000⟩	38.7	18.1	**43.2**	31	20.3	**48.7**	**41.5**	18.2	20.3
⟨001⟩	3.8	2.4	**93.8**	2.7	3.7	**93.6**	3.5	2.9	**93.6**
⟨010⟩	0.7	**95.5**	3.8	1.5	**93.6**	4.9	0.7	**94.5**	4.8
⟨011⟩	6.8	22.9	**70.3**	7.8	27.5	**64.7**	10	24	**66**
⟨100⟩	**97.6**	1.4	1	**98.8**	1.1	0.1	**98.9**	0.7	0.4
⟨101⟩	**71.9**	2.9	25.2	20.7	5.2	**74.1**	38.1	2.6	**59.3**
⟨110⟩	**86.5**	11.8	1.7	**59**	34.5	6.5	**57.4**	33.5	9.1
⟨111⟩	**56.7**	10.3	33	32.1	16.8	**51.1**	**45**	11.1	43.9

Table 2: For each label, percentage of blended segments generated using that label, that was classified as the different games. Highest percentage classification for each label-dimensionality pair highlighted in bold.

and only if its corresponding bit in the label is 0. These results suggest that segments generated using these labels do blend the games. Example blended levels are shown in Figure 7. We see that adding KI or MM labels to SMB segments makes them more vertical. Similarly, adding SMB labels to KI and MM gives them a more horizontal progression.

As further evaluation, for each blend label, we generated 1000 segments and computed the E-distance between them and the original training segments for each game. E-distance (Székely and Rizzo 2013) measures the similarity between two distributions and has been suggested as a suitable metric for comparing generative models (Summerville 2018). The lower the E-distance between two distributions, the more similar they are. Thus, for example, we would expect the E-distance between original SMB levels and those generated using the blend conditioning label ⟨100⟩ to be the lowest among all labels with the value being higher for labels not containing SMB (i.e. 0 in the first label element). For computing E-distance, we used four tile-based properties: *Density*, *Nonlinearity*, *Leniency* and *Interestingness* as described in Snodgrass and Sarkar (2020). Results for all 3 blend CVAE models are shown in Figure 8, averaged across the models. Results for SMB and MM are as expected with the lowest E-distance for all models being for labels ⟨100⟩ and ⟨001⟩. For these, E-distance is higher when that game is not included in the label than when it is, as expected. Comparing to SMB, we note the sharp drop as the first label bit flips to 1 indicating inclusion of SMB in the conditioning. Similarly, comparing to MM, we note the E-distance oscillate lower to higher as the rightmost bit switches between 1 and 0, indicating inclusion of MM. Interestingly, results for KI are not as expected with label ⟨010⟩ producing the second highest E-distance and would be worth exploring in the future. Overall however, the E-distance trends suggest that conditioning is able to generate levels of different blends.

Conclusion and Future Work

We explored conditional VAEs and how their affordances can inform level design applications by allowing designers to use labels to control level generation and blending as well as edit existing levels using different labels. There are several future avenues to consider. While CVAEs en-

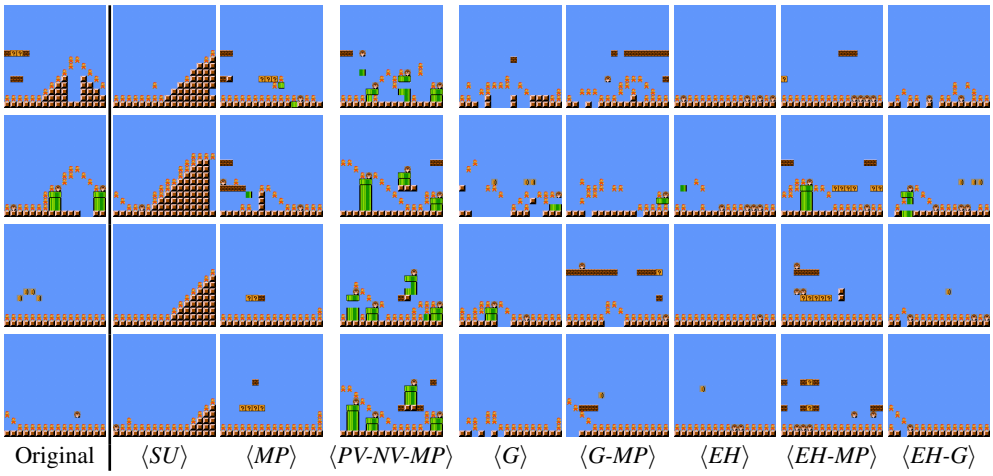

Figure 6: Example segments generated by conditioning on SMB design patterns. The first segment in each row is from the game. Every other segment in its row is generated using the same vector as the original but conditioned using the label for that column. Results shown were generated using the 32-dimensional model. Labels indicate design patterns as defined previously.

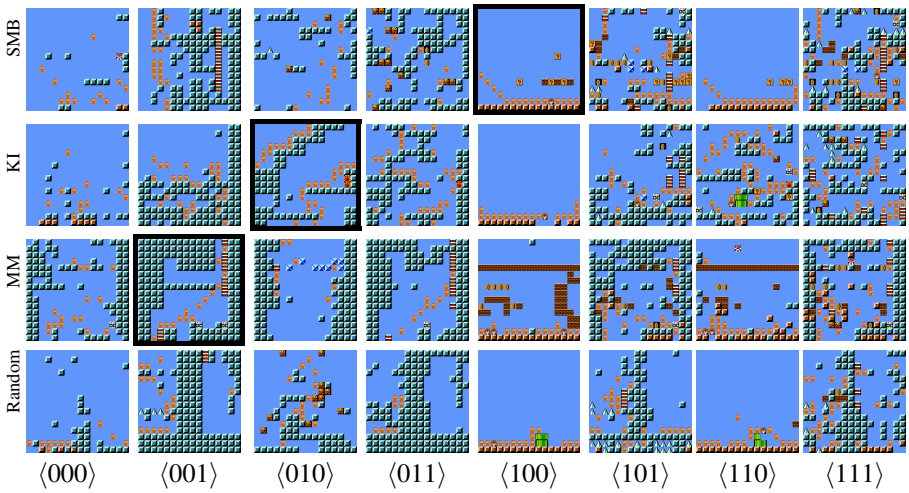

Figure 7: Segments generated by conditioning on blend labels, using an original segment from SMB (first row), KI (second) and MM (third) and a random segment (last). First, second and third elements of the label correspond to SMB, KI and MM respectively. Results were generated using the 128-dimensional model. For first 3 rows, bordered segments are originals.

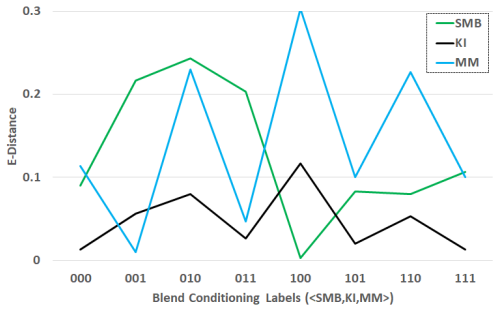

Figure 8: E-distances between original training distributions and distributions of 1000 levels generated using each of the blend conditioning labels.

able controllability without having to run evolution, the latter could still be useful, for example, to generate content using labels that are infrequent in the training data. This could also help blending applications. Currently, the CVAE generates blended levels using given labels but the actual blended content is not controllable. Evolution could search for levels that in addition to being conditioned by a specific label, optimize a desired objective. In this work, we highlighted different CVAE affordances for controllable PCG but hope to focus on each affordance more thoroughly in the future. A user evaluation for design pattern conditioning would be useful as would playability evaluations. Additionally, this approach worked with level segments. To generate whole levels, it could be combined with the approach in Sarkar and Cooper (2020a) that learns a sequential segment generation model where generated segments logically follow to create whole levels. Finally, we intend to incorporate such CVAE models into game design tools to enable conditional generation and design, as outlined in Sarkar and Cooper (2020b).

References

Alvarez, A.; Dahlskog, S.; Font, J.; and Togelius, J. 2020. Interactive constrained map-elites analysis and evaluation of the expressiveness of the feature dimensions. *arXiv preprint arXiv:2003.03377*.

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

Dahlskog, S., and Togelius, J. 2012. Patterns and procedural content generation: revisiting Mario in world 1 level 1. In *Proceedings of the 1st Workshop on Design Patterns in Games*, 1–8.

Dormans, J. 2010. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1–8.

Fauconnier, G., and Turner, M. 1998. Conceptual integration networks. *Cognitive Science* 22(2):133–187.

Goodfellow, I.; Abadie-Pouget, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Gow, J., and Corneli, J. 2015. Towards generating novel games using conceptual blending. In *Proceedings of the 11th Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

Guzdial, M., and Riedl, M. 2016a. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Guzdial, M., and Riedl, M. 2016b. Learning to blend computer game levels. In *Proceedings of the 7th International Conference on Computational Creativity*.

Kingma, D., and Welling, M. 2013. Auto-encoding variational Bayes. In *The 2nd International Conference on Learning Representations (ICLR)*.

Liapis, A.; Yannakakis, G.; and Togelius, J. 2013. Sentient Sketchbook: Computer-aided game level authoring. In *Proceedings of the 8th International Conference on the Foundations of Digital Games*.

Mirza, M., and Osindero, S. 2014. Conditional generative adversarial networks. *arXiv preprint arXiv:1411.1784*.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Sarkar, A., and Cooper, S. 2018. Blending levels from different games using LSTMs. In *AIIDE Workshop on Experimental AI in Games*.

Sarkar, A., and Cooper, S. 2020a. Sequential segment-based level generation and blending using variational autoencoders. In *Proceedings of the 11th Workshop on Procedural Content Generation in Games*.

Sarkar, A., and Cooper, S. 2020b. Towards game design via creative machine learning (GDCML). In *Proceedings of the IEEE Conference on Games*.

Sarkar, A.; Summerville, A.; Snodgrass, S.; Bentley, G.; and Osborn, J. 2020. Exploring level blending across platformers via paths and affordances. In *16th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Sarkar, A.; Yang, Z.; and Cooper, S. 2019. Controllable level blending between games using variational autoencoders. In *AIIDE Workshop on Experimental AI in Games*.

Schrum, J.; Gutierrez, J.; Volz, V.; Liu, J.; Lucas, S.; and Risi, S. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Genetic and Evolutionary Computation Conference (GECCO)*.

Smith, G.; Treanor, M.; Whitehead, J.; and Mateas, M. 2009. Rhythm-based level generation for 2D platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, 175–182.

Snodgrass, S., and Ontañón, S. 2016. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *25th International Joint Conference on Artificial Intelligence*.

Snodgrass, S., and Ontañón, S. 2017. Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games* 9(4):410–422.

Snodgrass, S., and Sarkar, A. 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*.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning structured output representation using deep conditional generative models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*.

Summerville, A., and Mateas, M. 2016. Super Mario as a string: Platformer level generation via LSTMs. *Proceedings of 1st International Joint Conference of DiGRA and FDG*.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The video game level corpus. In *7th Workshop on Procedural Content Generation at 1st Joint International Conference of DiGRA and FDG*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10(3):257–270.

Summerville, A. 2018. Expanding expressive range: Evaluation methodologies for procedural content generation. In *14th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Székely, G. J., and Rizzo, M. L. 2013. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference* 143(8):1249–1272.

Togelius, J.; Preuss, M.; Beume, N.; Wessing, S.; Hagelback, J.; and Yannakakis, G. N. 2010. Multiobjective exploration of the Starcraft map space. In *2010 IEEE Symposium on Computational Intelligence and Games (CIG)*.

Torrado, R. R.; Khalifa, A.; Green, M. C.; Justesen, N.; Risi, S.; and Togelius, J. 2019. Bootstrapping conditional GANs for video game level generation. *arXiv preprint arXiv:1910.01603*.

Valls-Vargas, J.; Zhu, J.; and Ontañón, S. 2017. Graph grammar-based controllable generation of puzzles for a learning game about parallel programming. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 1–10.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 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. ACM.

Yan, X.; Yang, J.; Sohn, K.; and Lee, H. 2015. Attribute2image: Conditional image generation from visual attributes. *arXiv preprint arXiv:1512.00570*.