Dungeon and Platformer Level Blending and Generation using Conditional VAEs

Anurag Sarkar
Northeastern University
Boston, MA, USA
sarkar.an@northeastern.edu

Seth Cooper
Northeastern University
Boston, MA, USA
se.cooper@northeastern.edu

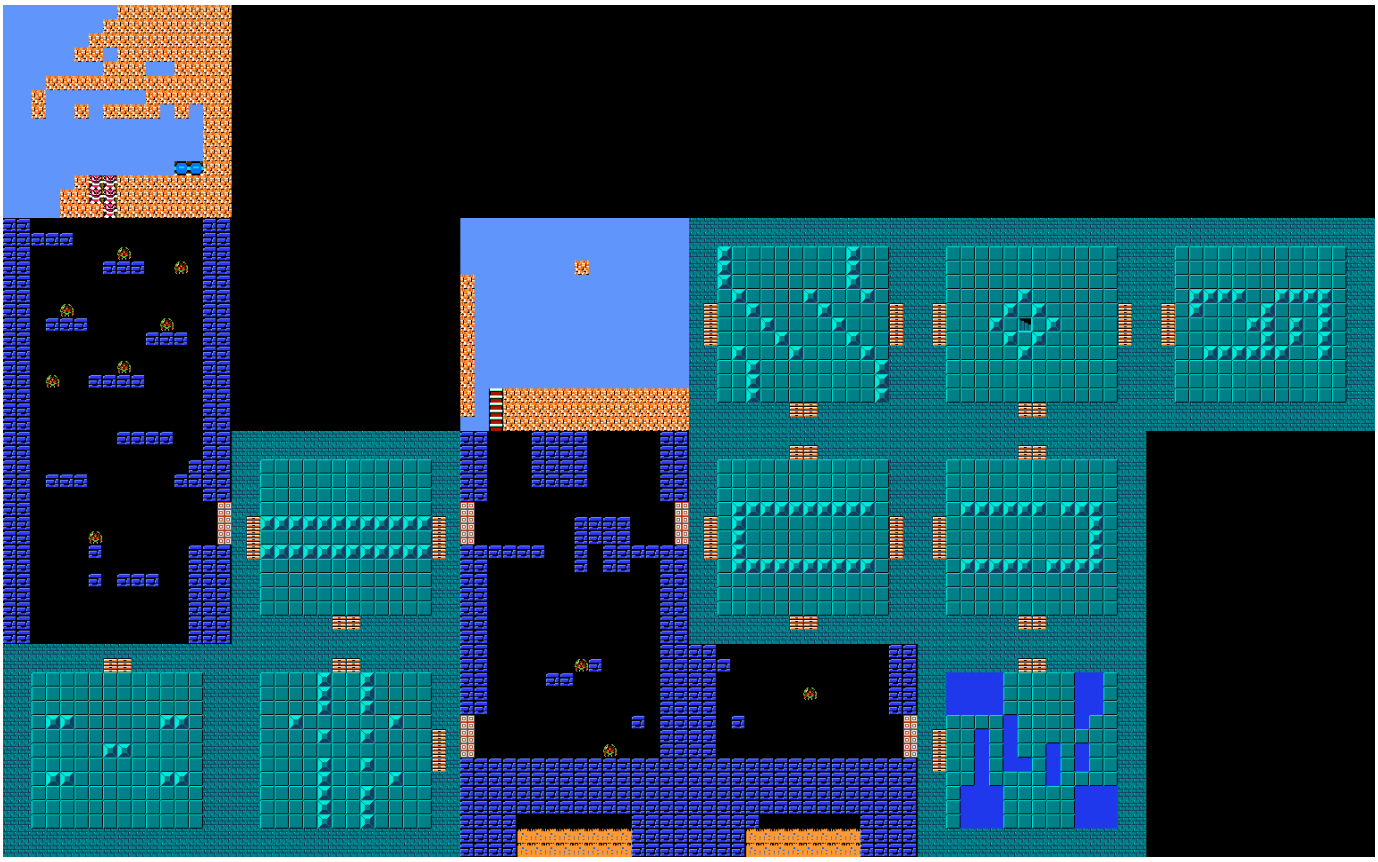


Fig. 1: Example blended Zelda–Metroid–Mega Man level

***Abstract*—Variational autoencoders (VAEs) have been used in prior works for generating and blending levels from different games. To add controllability to these models, conditional VAEs (CVAEs) were recently shown capable of generating output that can be modified using labels specifying desired content, albeit working with segments of levels and platformers exclusively. We expand these works by using CVAEs for generating whole platformer and dungeon levels, and blending levels across these genres. We show that CVAEs can reliably control door placement in dungeons and progression direction in platformer levels. Thus, by using appropriate labels, our approach can generate whole dungeons and platformer levels of interconnected rooms and segments respectively as well as levels that blend dungeons and platformers. We demonstrate our approach using *The Legend of Zelda, Metroid, Mega Man* and *Lode Runner*.**

***Index Terms*—PCGML, variational autoencoder, level generation, game blending**

I. Introduction

Among the many approaches for Procedural Content Generation via Machine Learning (PCGML) [1], variational autoencoders (VAEs) [2] have been used in several recent works for generating levels [3], [4] as well as blending levels across different games [5], [6]. VAEs consist of encoder and decoder networks that learn to map data to and from a continuous latent space respectively. Once trained, the latent space can be sampled to generate new data. While shown to be effective in generating and blending levels for several games, standard VAEs work with fixed size inputs/outputs, thus forcing models to train on and thereby generate level segments rather than whole levels, due to the dearth of training data for games. Additionally, VAEs enable generation via random sampling

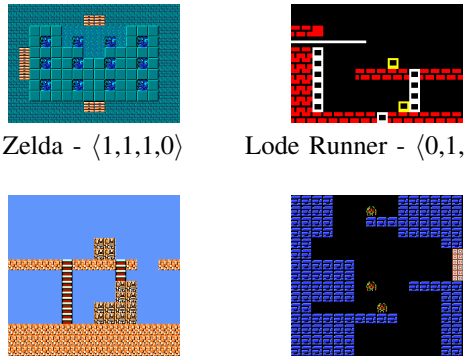


Fig. 2: Example segments with corresponding directional labels indicating doors/openings in ⟨Up,Down,Left,Right⟩.

of latent space vectors, thus methods such as latent variable evolution [7] have been used to add controllability by searching for latent vectors that satisfy certain properties. Alternatively, conditional VAEs [8] augment regular VAEs by allowing them to be trained using labeled data and thus enable the generation of outputs conditioned by specific labels. This obviates having to run evolutionary search post-training and enables controllability via the training process itself. Prior work [9] explored the use of CVAEs for PCGML, finding them capable of generating outputs with desired elements and patterns as well as blending outputs across games. However, this was limited to level segments and like much PCGML research, was restricted to the platformer domain.

In this paper, we expand on the above work by using conditional VAEs to generate entire levels and do so for multiple genres. Specifically, we show that by having the conditioning labels indicate the direction of door placement in dungeon rooms and progression in platformer level segments, we can generate rooms and segments with orientations that enable them to be reliably connected in order to form whole levels. Moreover, this also enables us to generate levels consisting of both dungeon rooms and platformer level segments, thus enabling blending levels from different genres unlike prior PCGML blending work which did so only across different platformers. We demonstrate our approach using levels from the platformers *Metroid* and *Mega Man*, the puzzle platformer *Lode Runner* and dungeons from *The Legend of Zelda*. Our work thus contributes, to our knowledge, 1) a new PCGML approach for generating whole levels using conditional VAEs and the first to be applied in multiple genres and 2) the first PCGML approach for blending levels across different genres.

II. Background

A significant body of PCGML research has used variational autoencoders (VAEs) [2] with most focusing on generating and blending platformer levels [3], [5], [6], [9]. Advanced models such as Gaussian Mixture VAEs [3] have also been used for injecting controllability into vanilla VAE models. Controllability has been the focus of several PCGML works involving GANs [10] and VAEs since random sampling of the latent space does not afford much control. A common approach for controllability is latent variable evolution [7] which utilizes evolution to look for latent vectors corresponding to desired content, as demonstrated for both GANs [11] and VAEs [5]. Relatedly, [12] enabled interactive exploration of a GAN latent space to evolve desired Mario levels and Zelda dungeons.

Unlike these hybrid approaches that add controllability via a method separate from training the model, conditional VAEs (CVAEs) [8] enable control as part of the model itself. When training CVAEs, each input is associated with a user-specified label. The encoder and decoder then learn to use these labels, and thus the information provided by them, to map inputs to and from the latent space respectively. Prior work [9] showed that CVAEs can generate levels conditioned on labels specifying the content and patterns desired to be present in the levels as well as blend levels from different games by having the label indicate the games to blend. We build on this by expanding beyond platformers and generating whole levels.

Generating whole levels is particularly challenging when working with latent models for two reasons. First, such models work with fixed size inputs and outputs, and second, the general dearth of training data for PCGML necessitating training on segments rather than whole levels to ensure enough training data. While generating whole levels by stitching together successively generated segments [13] is acceptable for a game like Mario with uniform progression, this does not work for games that progress in multiple directions and orientations. This was addressed in [14] via an approach combining a VAE-based sequential model and a classifier to generate and then place a segment relative to the previous one, thus generating whole levels by an iterative loop of decoding and encoding successive segments. However this afforded control only via definition of the initial segment with the orientation of successive segments and properties of the blend not being controllable. This issue was also addressed by [15] which focused on generating whole Mega Man levels by using multiple GANs to model levels with different orientations. Our work differs in that compared to [14], each segment and the blend properties are controllable and compared to [15], we use VAEs and model differences in orientations with separate labels rather than separate models. Additionally, we extend beyond the platformer domain.

While most PCGML works have focused on side-scrolling platformers, a handful have worked with other genres. Doom levels have been generated using GANs [16] while Lode Runner levels have been generated using GANs [17], Markov models [18] and VAEs [4]. Both [19] and [20] generated Zelda dungeons using Bayes nets. Similar to our use of latent models to create whole dungeons out of generated rooms, [21] used an approach that utilized a graph grammar to generate a dungeon layout and a GAN to generate rooms to fill that layout. Our work differs in using VAEs and that the properties of dungeon rooms are determined by conditioning rather than a grammar.

Game blending was proposed by [22] and refers to generating new games by blending levels and/or mechanics of existing games. Since then, many works [5], [6], [9], [23] have used VAEs for blending levels across several platformers though

only working with segments. The approach in [14] generated whole levels that are blended but without affording control over blend contents. The method presented in this paper allows generating whole level blends in a controllable manner in addition to blending games from different genres. In doing so, this work constitutes another PCGML approach under combinational creativity [24], similar to recent works looking at domain transfer [25] and automated game generation [26], all of which blend and/or combine existing domains to generate content for new ones. Additionally, in being a creative ML approach for level generation and blending, this work can also be categorized under the recently defined Game Design via Creative Machine Learning (GDCML) [27] set of approaches.

III. Method

A. Level Representation and Conditioning

We used levels from the action-adventure game *The Legend of Zelda* (Zelda), the action platformers *Metroid* and *Mega Man* (MM), and the puzzle platformer *Lode Runner* (LR), thus spanning several genres and types of games. Levels were taken from the Video Game Level Corpus (VGLC) [28] and were in text format with different characters mapping to different in-game tiles. The premise that facilitates the use of CVAEs is that levels in these games are composed of segments connected together in a meaningful way to enable gameplay. This is most evident in Zelda where discrete rooms are connected based on the location of doors but is also true for platformers where levels can be considered to consist of discrete segments connected based on the directionality of player movement through them. Thus, training a CVAE on the discrete segments while using labels indicating how they can be connected to other segments should yield a model that can generate whole levels composed of these interconnected segments. For each game, labels were binary-encoded vectors of length 4 with each element corresponding to a direction with 1/0 indicating that progression in that direction was possible/not possible. For all games, the order was Up, Down, Left, Right, thus, a label of $\langle$1,0,0,1$\rangle$ indicated that progression was possible in the upward and rightward, but not in the downward or leftward, direction for the corresponding segment.

1) The Legend of Zelda: Training data for Zelda consisted of the discrete rooms taken from each dungeon in the VGLC. Rooms in Zelda are 11x16 tiles consisting of a 2-tile perimeter of wall tiles and a 7x12 playing area. We augmented the training set by adding horizontally and vertically flipped versions of each room, if not already present, finally ending up with a total of 502 rooms for training. Labels for each room were determined based on the location of the door tiles.

2) Metroid: For Metroid, we extracted 15x16 segments from the whole levels. Though Metroid levels are not composed of independent rooms like Zelda dungeons, we could still extract discrete segments based on the fact that horizontal segments of Metroid are 15 tiles high and vertical segments are 16 tiles wide. After filtering out segments that consisted entirely of block characters, we obtained 414 segments. Labels for each segment were assigned manually based on visual inspection and indicated in which directions that segment was open i.e. in which directions could a player enter or exit that segment. Doors in Metroid were considered an open direction.

3) Mega Man: Horizontal and vertical sections of MM have similar dimensions as in Metroid so here too, we extracted 15x16 segments, obtaining 143 for training. Labels were assigned similarly to how they were for Metroid segments.

4) Lode Runner: All LR levels have a fixed dimension of 22x32 tiles. Thus, a standard VAE is perfectly suited to generate whole levels but we still wanted to test our CVAE approach of building levels using labeled segments with LR and thus divided each of the 150 LR levels in the VGLC into 4 segments of 11x16 tiles, obtaining a total of 600 segments for training. Labels for each segment were assigned to indicate in which direction that segment was connected to another e.g. the top-left segment was assigned $\langle$0,1,0,1$\rangle$ i.e. Down and Right, since it had a segment to the right and a segment below it.

Unlike prior work using VAEs, rather than extract segments with redundancy from platformer levels by sliding a window one column/row at a time, we used non-overlapping segments to be more compatible with the discrete Zelda rooms as well as for directional labels to be more meaningful and avoid situations such as e.g., Metroid segments with doors in the middle. Example segments and labels are shown in Figure 2.

B. Game and Genre Blending

In addition to working with the games separately, we wanted to test the CVAE in generating levels blended across different games. Since the labels indicate directions in which progress is possible, irrespective of whether they are doors in dungeon rooms or pathways/openings in platformers, a model trained on similarly labeled sections from multiple games taken together might be able to produce blended levels composed of such interconnected sections. For example, a model trained on Zelda and Metroid should be able to produce blended levels composed of Zelda rooms connected with Metroid segments and blended segments consisting of Zelda and Metroid elements, thus giving us an approach for potentially blending levels across not just games but different genres. For these CVAE models, we extended the labels to also specify the games being blended i.e. for a model trained on segments/rooms from n games, we concatenated an n-element binary vector to the 4-element binary vector indicating the directionality as before e.g. when blending two games, labels for segments from game 1 were concatenated with $\langle$1,0$\rangle$ while those from game 2 were concatenated with $\langle$0,1$\rangle$. We trained blended models of *Metroid–Mega Man*, *Zelda–Lode Runner*, *Zelda–Metroid*, *Zelda–Mega Man* and *Zelda–Mega Man–Metroid*.

Training a blended model on Metroid and MM was straightforward since both consist of 15x16 segments. Similarly, 11x16 Zelda rooms were compatible with 11x16 LR segments. For blending Zelda with Metroid and MM, we padded the Zelda rooms by duplicating each room's two outermost floor rows both to the north and to the south, thus yielding 15x16 rooms. Being discrete and independent, Zelda rooms were well-suited to being padded. Padding LR segments was less

resistant to impacting directionality. Thus for that and to keep to a manageable number of models, we blended LR with only Zelda. For the 3-game blend, game labels were ⟨0,0,1⟩, ⟨0,1,0⟩ and ⟨1,0,0⟩ referring to MM, Metroid and Zelda respectively.

Besides using models trained on multiple games taken together, blending can also be done by taking turns generating level sections using multiple single-game models. This was explored in [29] using separate LSTM models of Mario and *Kid Icarus*. Since our CVAEs use the same underlying definition for labels (i.e. directionality), given a set of labels, we can use different models to generate different level sections to obtain a level made of segments from multiple games, as in Figure 1. One could select which game model to use based on specified probabilities which enables generating levels blending desired proportions of games. Since this uses the single-game models, we do not separately evaluate this approach.

C. Layout Algorithm

We use a simple approach to generate layouts within which to place segments. We start by placing a single segment with closed labels on all four sides. At each step, we select a random side of a segment labeled as closed. If there is no segment next to that side, a segment is placed there with all sides labeled closed. We then connect the originally selected segment to the (pre-existing or newly placed) segment next to the selected side by setting the label to open on the adjoining sides. The number of steps to run is chosen randomly from a small range. We combine this with the CVAE by looping through each layout segment, determining the appropriate label based on the directions in which that segment is open/closed and then use that to condition the generation of a sampled latent vector to place in that location. For blending, any provided game label could be concatenated to the determined direction label to produce a segment. For the blend examples shown in the paper, we randomly set the game label bits to 1 or 0 for each game. This method could also accept probabilities for each game and then generate levels that blend the games accordingly.

D. Modeling and Conditional Training/Generation

We trained a CVAE model for each of the above games and blends. The encoder and decoder for all models consisted of 4 fully connected layers each, using ReLU activation. Models were trained for 10000 epochs using the Adam optimizer and a learning rate of 0.001, decayed by 0.01 every 2500 epochs. These hyperparameters were set based on the prior VAE-based PCGML work. All models were trained using PyTorch [30]. To study different latent sizes, we trained 4 versions of each model using latent dimensions of 4, 8, 16 and 32.

Training a CVAE involves associating a label with each input, concatenating them together and forwarding them through the encoder to obtain a latent vector which is then concatenated with the same label and forwarded through the decoder. Through training, the encoder and decoder thus learn to use the label to encode inputs and generate outputs respectively. After training, generation is done by sampling a vector from the latent space, concatenating it with the desired label and forwarding it through the decoder to obtain the output. More technical details regarding CVAEs can be found in [8], [31].

IV. Evaluations and Discussion

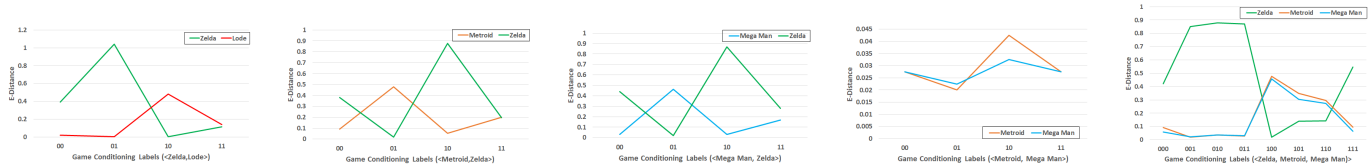
A. Directional Labeling Accuracy

For evaluations, we were primarily interested in testing the accuracy of labeling, i.e. if the segments generated by the CVAEs were true to the directionality/orientation as indicated by the labels used to generate them. For Zelda, this is straightforward to verify since we just check the location of door tiles in the generated segments, similar to how the training segments were labeled. For each of the 3 other games, we trained a random forest classifier on their training segments using 10-fold cross validation and the directionality labels as the class labels with Metroid, MM and LR having 12, 9 and 4 unique classes/directionalities respectively. All classifiers were implemented using scikit-learn [32]. We obtained mean classification accuracies of 98%, 81% and 51% respectively. The low accuracy for LR is likely due to training on level quadrants and that structure and progression doesn't particularly vary across them since original levels do not vary gameplay based on quadrants. We note however that this implies that different 11x16 LR segments are interchangeable in terms of being able to be combined with respect to directionality and progression. To evaluate conditioning accuracy, for a generated segment, we compared the label predicted by the classifier with the one used to condition the segment's generation. While ideally we want exact matches between predicted and conditioning labels, i.e. the generated segment to be open and closed in the exact label-specified directions, it is sufficient that the generated segment has just the desired open directions, regardless of whether the desired closed directions end up being open in the generated segment. We want to avoid the reverse case i.e. desired open directions being closed in the generated segment. Thus, in addition to *exact* label matches, we were interested in matches where the bits set to 1 in the conditioning label were also set to 1 in the predicted label, regardless of the value of the other bits. We refer to these as *admissible* matches. Note that by definition, an exact match is also admissible. An additional point of note is that not all possible directional labels are present in each game. Since the directional labels are binary vectors of length 4, there are $2^4 = 16$ possible labels. However, as mentioned above, only 12, 9 and 4 of these are present in the training data of Metroid, MM and LR respectively, with only Zelda having all 16. It is thus reasonable to expect that the models will do a better job of generating matching segments when using labels that appear in the training data compared to those that do not. Thus, except for Zelda, we track matches separately for IN-game and OUT-of-game labels.

For each model, we sampled 1000 latent vectors at random and used each of the 16 possible labels for conditioning, thus generating 16000 segments. For each segment, we applied the relevant classifier and compared the predicted label with the label used for conditioning, separately computing the percentage of *exact* and *admissible* matches for IN and OUT labels. That is, we computed what percentage of IN labels

	Zelda		Metroid			Mega Man			Lode Runner		
Latent	Exact	Admissible	Exact-IN	Admissible-IN	Admissible-OUT	Exact-IN	Admissible-IN	Admissible-OUT	Exact-IN	Admissible-IN	Admissible-OUT
4	99.81	99.84	**69.52**	90.96	71.15	**61.9**	88.81	27.9	**95.48**	**95.48**	39.36
8	**99.96**	**99.96**	69.28	**91.29**	**73.33**	61.62	**89.4**	27.87	94.78	94.78	39.97
16	99.85	99.91	67.63	90.38	72.13	58.9	86.57	28.14	94.78	94.78	39.9
32	99.95	**99.96**	62.34	89.33	64.6	53.78	83.21	**28.99**	94.78	94.78	**40.18**

TABLE I: Directional label accuracy.



(a) Zelda–Lode Runner (b) Zelda–Metroid (c) Zelda–Mega Man (d) Metroid–Mega Man (e) Zelda–Met–MM

Fig. 3: E-Distances for different game blends.

⟨Zelda,LR⟩	4		8		16		32	
	Z	LR	Z	LR	Z	LR	Z	LR
⟨0,0⟩	48.5	**51.5**	**51.6**	48.4	**52.2**	47.8	**62.1**	37.9
⟨0,1⟩	0	**100**	0.2	**99.8**	0	**100**	1	**99**
⟨1,0⟩	**100**	0	**100**	0	**100**	0	**100**	0
⟨1,1⟩	**74.9**	25.1	**95.1**	4.9	**88.7**	11.3	**67.3**	32.7

⟨Metroid,MM⟩	4		8		16		32	
	Met	MM	Met	MM	Met	MM	Met	MM
⟨0,0⟩	**53.9**	46.1	**50.7**	49.3	**54.1**	45.9	48.2	**51.8**
⟨0,1⟩	1.4	**98.6**	3.1	**96.9**	5.1	**94.9**	4.1	**95.9**
⟨1,0⟩	**96.7**	3.3	**94.3**	5.7	**92**	8	**92.7**	7.3
⟨1,1⟩	37.3	**62.7**	46.2	**53.8**	44	**56**	49.8	**50.2**

⟨Metroid,Zelda⟩	4		8		16		32	
	Met	Z	Met	Z	Met	Z	Met	Z
⟨0,0⟩	**66.9**	33.1	35.9	**64.1**	**53.9**	46.1	47.6	**52.4**
⟨0,1⟩	0	**100**	0	**100**	0	**100**	0	**100**
⟨1,0⟩	**99.4**	0.6	**99.7**	0.3	**100**	0	**99.8**	0.2
⟨1,1⟩	38.1	**61.9**	31.8	**68.2**	16	**84**	25.9	**74.1**

⟨MM,Zelda⟩	4		8		16		32	
	MM	Z	MM	Z	MM	Z	MM	Z
⟨0,0⟩	**64.8**	35.2	38.3	**61.7**	**54.9**	45.1	**61.3**	38.7
⟨0,1⟩	0	**100**	0	**100**	0	**100**	0	**100**
⟨1,0⟩	**98.8**	1.2	**99.9**	0.1	**100**	0	**99.3**	0.7
⟨1,1⟩	10.8	**89.2**	24.7	**75.3**	41.6	**58.4**	**71.1**	28.9

TABLE II: Blend accuracy for all two-game blends.

⟨Zelda,	4			8			16			32		
Met,MM⟩	Zel	Met	MM	Zel	Met	MM	Zel	Met	MM	Zel	Met	MM
⟨0,0,0⟩	**54.1**	23.3	22.6	**38.2**	33.1	28.7	34.8	**36.3**	28.9	**54.2**	25.3	20.5
⟨0,0,1⟩	0.3	8.5	**91.2**	2.4	5.8	91.8	1	9.9	**89.1**	0.1	13.3	**86.6**
⟨0,1,0⟩	0.1	**94.1**	5.8	0	91.3	8.7	0.2	**94**	5.8	0	**89.1**	10.9
⟨0,1,1⟩	0.1	41.1	**58.8**	0.1	38.8	**61.1**	0	47.2	**52.8**	0	46.9	**53.1**
⟨1,0,0⟩	**100**	0	0	**100**	0	0	**100**	0	0	**100**	0	0
⟨1,0,1⟩	**91.2**	0.5	8.3	**89.8**	0.3	9.9	**98.2**	0.3	1.5	**51.3**	6.1	42.6
⟨1,1,0⟩	**75**	21.6	3.4	**90.9**	8.7	0.4	**96.8**	3	0.2	**61.8**	31.3	6.9
⟨1,1,1⟩	23.6	32.9	**43.5**	44.4	21.1	**34.5**	**48.2**	24.7	27.1	6.5	33.3	**60.2**

TABLE III: Blend Accuracy for Zelda–Metroid–Mega Man.

yielded *exact* and *admissible* matches and what percentage of OUT labels yielded such matches. Results are shown in Table I. All Zelda models achieved near 100% accuracy, suggesting that door placement using conditioning is very reliable for Zelda rooms. The *exact* match percentages for OUT labels were 0% in all non-Zelda games so we omitted them. The

Zelda	Original	4	8	16	32
Density	0.59 ± 0.08	0.58 ± 0.06	0.58 ± 0.06	$\mathbf{0.59 \pm 0.06}$	0.59 ± 0.07
Symmetry	0.76 ± 0.06	0.76 ± 0.04	0.76 ± 0.04	0.76 ± 0.05	0.75 ± 0.05
Metroid	Original	4	8	16	32
Density	0.39 ± 0.16	$\mathbf{0.41 \pm 0.09}$	$\mathbf{0.4 \pm 0.08}$	$\mathbf{0.41 \pm 0.08}$	$\mathbf{0.4 \pm 0.08}$
Symmetry	0.67 ± 0.12	$\mathbf{0.59 \pm 0.04}$	$\mathbf{0.6 \pm 0.05}$	$\mathbf{0.6 \pm 0.05}$	$\mathbf{0.61 \pm 0.05}$
Mega Man	Original	4	8	16	32
Density	0.38 ± 0.18	0.4 ± 0.13	0.41 ± 0.13	0.39 ± 0.14	0.39 ± 0.13
Symmetry	0.65 ± 0.13	0.62 ± 0.08	0.63 ± 0.07	0.64 ± 0.09	0.63 ± 0.07
Lode	Original	4	8	16	32
Density	0.41 ± 0.17	0.39 ± 0.14	0.39 ± 0.15	$\mathbf{0.38 \pm 0.15}$	0.39 ± 0.15
Symmetry	0.52 ± 0.14	0.51 ± 0.1	0.53 ± 0.1	$\mathbf{0.53 \pm 0.11}$	$\mathbf{0.53 \pm 0.11}$

TABLE IV: Means $\pm$ std. devs. for *Density* and *Symmetry* of generated segments. Values in bold were significantly different from the original segments.

Latent	Zelda	Metroid			Mega Man			Lode Runner		
	Ovr	IN	OUT	Ovr	IN	OUT	Ovr	IN	OUT	Ovr
4	43.1	**59.9**	**88.8**	**67.2**	69.8	**95.9**	81.5	78.4	**99.5**	94.2
8	45.2	58.1	87.3	66.6	**72.7**	95.4	**83.3**	82.6	99.2	94.7
16	45.8	58.4	86.6	65.5	69.9	93.6	80	**84.2**	98.8	**95.8**
32	**48.3**	58.6	77.1	63.6	71.9	89.1	79.3	84.1	98.9	95.5

TABLE V: Novelty of generated segments. *Ovr* is the overall percentage, combining both IN and OUT label percentages.

next highest percentages are for LR though only 4 out of 16 labels are IN labels so for the remaining 12, LR models produced *admissible* matches only about 40% of the time. Metroid outperformed MM in all 3 types of matches. Overall, in all cases, we observed about 90% *admissible* matches when using IN labels suggesting that the models can very reliably produce openings/doors in desired directions as prescribed by the label, even if *exact* matches are not as frequent.

B. Blend Labeling Accuracy

Similarly, we wanted to evaluate the blend labeling accuracy. For each blended model, we trained a random forest classifier with 10-fold cross validation, using the game that a training segment belongs to as the class label. The classifier for the *Zelda–Metroid–Mega Man* blend worked with 3 class labels while the others all worked with two. For all blends involving Zelda, we obtained classification accuracies of 100% and an accuracy of 98% for the Metroid-Mega Man blend.

For each model, we sampled 100 latents and used each possible game+directional label to condition their generation. For 2-game blends, there were $2^{(2+4)}$=64 such labels, resulting in a total of 6400 generated segments and for the 3-game blend, there were $2^{(3+4)}$=128 possible labels, resulting in 12800 generated segments. In both cases, we applied the relevant blend classifier on each generated segment and compared the predicted label with the game label used for generation. In all cases, for each possible game label, we computed the percentage of times each of the games in that blend was predicted. We expect that when using single game labels (e.g. $\langle$0,1$\rangle$, $\langle$1,0$\rangle$) to generate segments, predictions will be high for that particular game. Also, when using blended game labels (e.g. $\langle$1,1$\rangle$, $\langle$1,0,1$\rangle$), predictions should be more spread out since here segments would blend the properties of the original games. Results are given in Tables II and III and are true to expectation. In all cases, when using single-game blend labels, the classifier predicts the relevant game close to 100% of the time for 2-game blends involving Zelda and at least 92.7% of the time for the Metroid-MM blend. In the 3-game blend this drops slightly to a minimum of 86.6% but is still mostly a very high percentage in all cases. Also, blend labels in the 2-game cases ($\langle$0,0$\rangle$, $\langle$1,1$\rangle$) lead to more spread out predictions with behavior varying across models, though in no clear pattern. There is more of a spread when using $\langle$0,0$\rangle$ than $\langle$1,1$\rangle$ with the latter in some cases causing one of the games to dominate. For the 3-game case, we also note more spread out predictions for the blend labels and note that when a game's label element is set to 0, that game is mostly predicted less than 1% of the time, except in the $\langle$0,0,0$\rangle$ case. Overall, these results, combined with those for the directional labels, suggest that conditioning can successfully generate segments with desired orientations as well as blend desired games together.

C. Tile-based Metrics and Novelty

We also evaluated the actual content of the generated segments. For this, we used the following two tile-based metrics:

- *Density* - proportion of a segment/room that is walls, ground/breakable tiles, doors, platforms and blocks.
- *Symmetry* - measure of the combined vertical and horizontal symmetry of a segment/room, computed by comparing pairs of rows and pairs of columns moving outward from the center and summing up the number of positions with the same tile.

Both measures were normalized to be between 0 and 1 with 1 representing a fully dense segment and a perfectly symmetrical segment for *Density* and *Symmetry* respectively.

For each game, we computed these values for each training segment. For evaluation, we sampled a number of latent vectors equal to the number of training segments for that game, and for each vector, computed the mean *Density* and *Symmetry*, averaged across segments generated by conditioning that vector using that game's IN directional labels. For a meaningful comparison between generated and training segments, we opted not to use the OUT labels. For each game, we compared the densities and symmetries of the

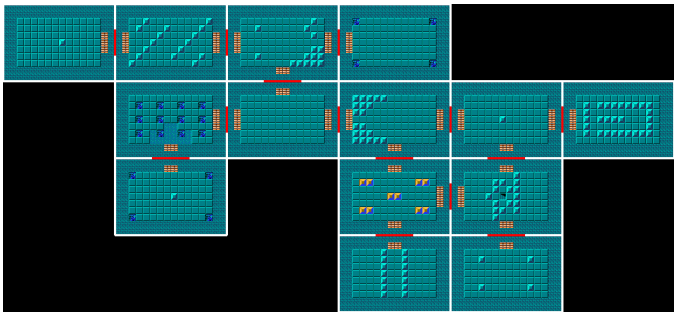


Fig. 4: Example Zelda level.

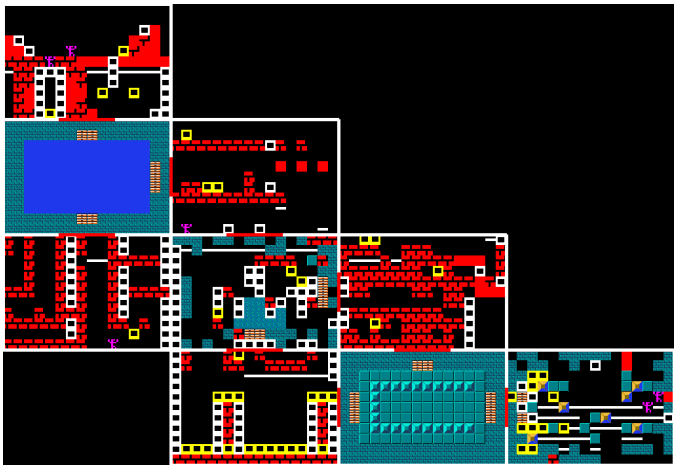


Fig. 5: Example blended Zelda–Lode Runner level.

training segments with those of the generated segments using a Wilcoxon Rank Sum test. Results are shown in Table IV. We observed significant differences (shown in bold) between training and generated segments ($p < .05$) for none of the values for MM, for the 16-dimensional Zelda model in terms of *Density*, for the 16-dimensional LR model in terms of *Density* and the 32-dimensional LR model in terms of both *Density* and *Symmetry* and in terms of all values for Metroid. The absence of statistical significance suggests models being able to successfully capture the structural patterns of the original games, in terms of the measured variables. Thus, the CVAEs for Zelda and MM did the best in terms of modeling the original levels. For LR, the bigger models fared worse than the smaller ones and overall, CVAEs for Metroid did worst in terms of closely capturing the original levels. However, note that the observed mean values for both *Density* and *Symmetry* are close to those of the original values with the standard deviation being far lower for the generated segments. This suggests that Metroid models were able to capture the primary structures and patterns of the original levels but failed in learning an adequate amount of variety and outlier features.

Since most models did well in capturing the structural patterns of the original levels, we wanted to check the performance of the models in terms of generating novel segments. For each game, we sampled 1000 latent vectors and generated

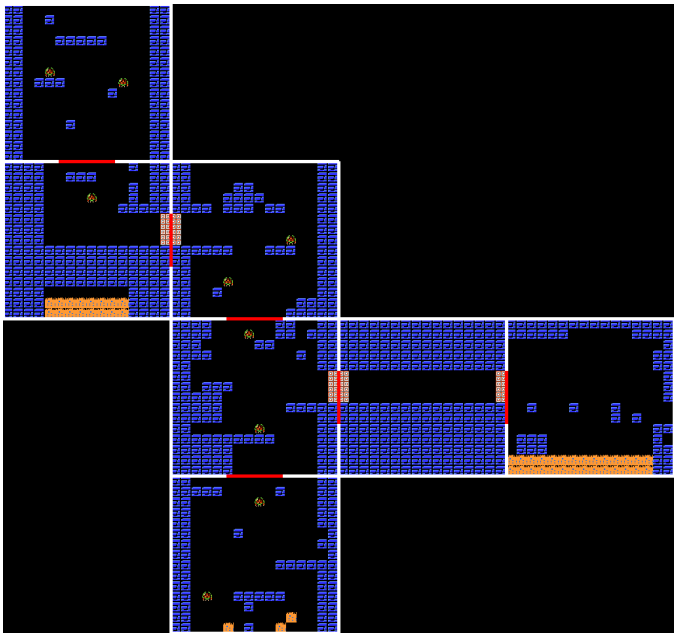


Fig. 6: Example Metroid level.

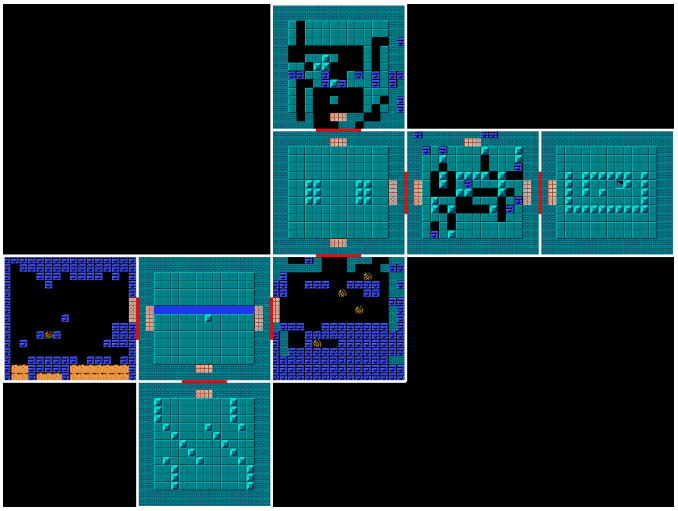


Fig. 7: Example blended Zelda–Metroid level.

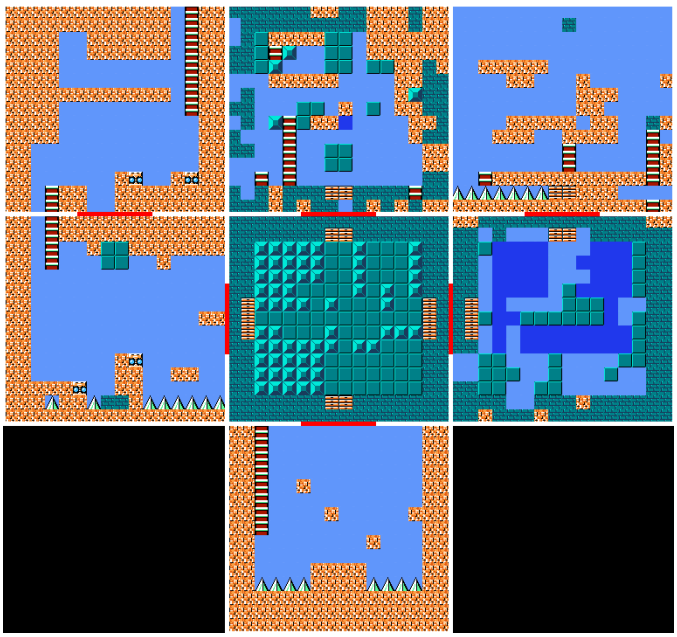


Fig. 8: Example blended Zelda–Mega Man level.

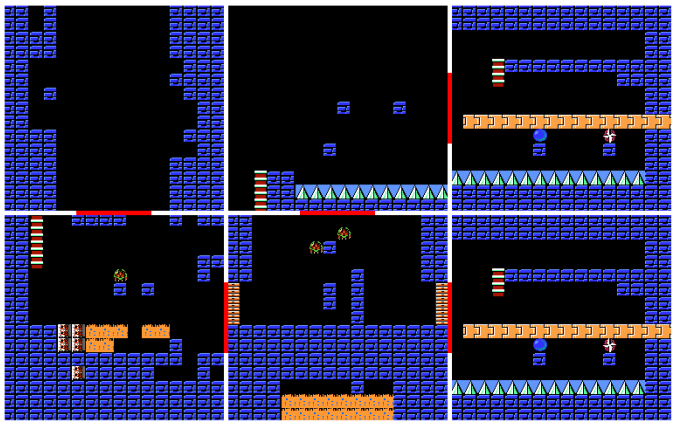


Fig. 9: Example blended Metroid–Mega Man level.

segments by conditioning each using all possible directional labels. We then computed what percentage of these segments was not in the original training set, computing separately for IN and OUT labels since we expect more of the IN label-generated segments to be in the training set. Results for this are given in Table V. The Zelda models produced the least novelty with over half the samples being those from the training set. LR models produced the highest variety, followed by MM, followed by Metroid, producing approximately 80%, 70% and 60% novel segments respectively when using the IN labels. Novelty was higher when using OUT labels but most of these segments likely have incorrect orientations as seen previously. Combining with the prior evaluation, the metrics for Zelda models being closest to training data can be explained by these models generating training segments about half of the time. This can be viewed as good or bad depending on if one prefers generated levels to closely resemble training levels or be more varied. Metroid having the lowest novelty out of the other 3 games but being the farthest from the training data in the prior evaluation reinforces the possibility that the models learned a specific subset of the training data and thus produce those segments when sampled but failed to learn more variations. Thus, future work should look at improving these Metroid models via more game-specific tuning and architectures.

Finally, to evaluate the segments generated by the blended models, we compared the distribution of training segments of the different games with the distribution obtained by using different game conditioning labels. For this comparison, we computed the E-distance between these two sets of segments. E-distance [33] is a measure of similarity between two distributions with lower values indicating higher similarity and

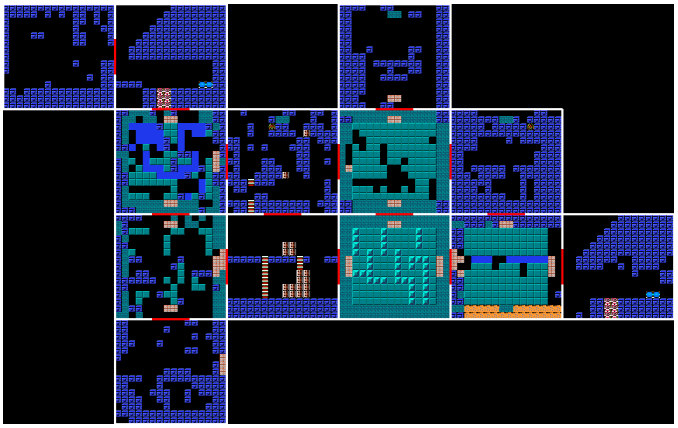


Fig. 10: Example blended Zelda–Metroid–Mega Man level.

has been previously recommended for evaluating generative models [34]. We used *Density* and *Symmetry* as defined previously for computing E-distance. The idea behind this evaluation is that we expect that, for example, for the Zelda–LR blend, the E-distance between the original Zelda segments and segments generated using the Zelda game label to be lower than the E-distance from segments generated using the LR game label and vice-versa. Results are shown in Figure 3. Except for Metroid-MM, all results are to expectation. When a game is specified in the label, the resulting distribution of generated segments is closer to the training distribution than when it is not. As an example, for Zelda–LR (fig 3.a), E-distance compared to the original LR levels (red line) is lowest when the label is $\langle 0,1 \rangle$ and highest when it is $\langle 1,0 \rangle$, with similar results for Zelda and the other blends. Results weren't as expected for the Metroid–MM blends however. This may be explained by the *Density* and *Symmetry* values being very similar for the training segments in both games, leading to similar E-distances. Overall, though, we can conclude based on this and the blend label evaluation, that blending in general can be controlled reliably by using these conditioning labels.

D. Visual Inspection

Example levels are shown in Figures 4-10. Segments are delineated using white borders with red dashes indicating doors/openings. Non-blended levels (Figures 4 and 6) look a lot more playable while blended levels are less so but arguably more interesting, consisting of smooth blended regions like the Metroid-Zelda sections in Figures 7 and 10 as well as more chaotic regions in Figure 5, blending ropes and ladders with doors and dungeons. As-is, this system could be used for generating blended levels for a companion system tasked with generating mechanics that render the levels playable. For e.g., what set of mechanics could one come up with to traverse Zelda dungeons while also being able to execute Metroid jumps and climbing ladders to collect gold? What mechanics make sense in regions that blend Zelda and Metroid? Answering such questions could help move us towards building systems capable of generating entirely new game designs.

V. Conclusion and Future Work

We presented an approach for generating whole platformer levels and dungeons by using CVAEs to control the directionality of generated segments and rooms, in turn also enabling us to take a step towards blending games from different genres, thus opening up several avenues to pursue in the future.

The main limitation of this work is the lack of playability evaluations. This was confounded by several factors. For Zelda, the VGLC dungeons contain neither lock and key tiles nor the triforce, all of which are crucial for playability. Thus, we restricted ourselves to noting that for Zelda, we obtained near-perfect reliability of door placement, ensuring that generated dungeons are at least traversable from room to room. Regarding Lode Runner, a player completes a level when they've collected all of the gold. However, since we worked with segments, this is harder to check since a segment may be playable but not have gold. Above all, blended levels would warrant new mechanics to be playable and generating mechanics for blended games is a problem we wish to address in the future. The next step is thus to generate fully playable versions of these dungeon-platformer hybrids and blends. We could take levels generated by this work and use a separate process to generate new/blended mechanics to be able to play them. Alternatively, we could use an agent capable of certain mechanics and try repairing a given set of generated levels, using an approach similar to [35]. It would also be fruitful to incorporate this work into a mixed-initiative or automated design tool where users can supply the labels as well as the proportions for blending. Finally, using different latent sizes did not have much impact both in terms of quantitative results and visual inspection. In the future, we'd like to focus specifically on investigating the effect of different latent sizes.

References

[1] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (PCGML)," *IEEE Transactions on Games*, 2018.

[2] D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *The 2nd International Conference on Learning Representations*, 2013.

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

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

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

[6] A. Sarkar, A. Summerville, S. Snodgrass, G. Bentley, and J. Osborn, "Exploring level blending across platformers via paths and affordances," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.

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

[8] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *NeurIPS*, 2015.

[9] A. Sarkar, Z. Yang, and S. Cooper, "Conditional level generation and game blending," in *EXAG Workshop*, 2020.

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

[11] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *GECCO*, 2018.

[12] J. Schrum, J. Gutierrez, V. Volz, J. Liu, S. Lucas, and S. Risi, "Interactive evolution and exploration within latent level-design space of generative adversarial networks," in *GECCO*, 2020.

[13] M. Green, L. Mugrai, A. Khalifa, and J. Togelius, "Mario level generation from mechanics using scene stitching," in *IEEE Conference on Games (CoG)*, 2020.

[14] A. Sarkar and S. Cooper, "Sequential segment-based level generation and blending using variational autoencoders," in *Foundations of Digital Games*, 2020.

[15] B. Capps and J. Schrum, "Using multiple generative adversarial networks to build better-connected levels for mega man," *arXiv preprint arXiv:2102.00337*, 2021.

[16] E. Giacomello, P. Lanzi, and D. Loiacono, "Doom level generation using generative adversarial networks," in *IEEE Games*, 2018.

[17] K. Steckel and J. Schrum, "Illuminating the space of beatable lode runner levels produced by various generative adversarial networks," *arXiv preprint arXiv:2101.07868*, 2021.

[18] S. Snodgrass and S. Ontañón, "Learning to generate video game maps using Markov models," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[19] A. Summerville and M. Mateas, "Sampling hyrule: Sampling probabilistic machine learning for level generation," 2015.

[20] A. Summerville, M. Behrooz, M. Mateas, and A. Jhala, "The learning of zelda: Data-driven learning of level topology," in *FDG workshop on Procedural Content Generation in Games*, 2015.

[21] J. Gutierrez and J. Schrum, "Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda," in *IEEE Congress on Evolutionary Computation (CEC)*, 2020.

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

[23] A. Sarkar and S. Cooper, "Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder," *arXiv preprint arXiv:2102.12463*, 2021.

[24] M. A. Boden, "The creative mind: Myths and mechanisms," 2004.

[25] S. Snodgrass and A. Sarkar, "Multi-domain level generation and blending with sketches via example-driven bsp and variational autoencoders," in *Foundations of Digital Games*, 2020.

[26] M. Guzdial and M. Riedl, "Conceptual game expansion," *IEEE Transactions on Games*, 2021.

[27] A. Sarkar and S. Cooper, "Towards game design via creative machine learning (gdcml)," in *IEEE Conference on Games (CoG)*, 2020.

[28] A. Summerville, S. Snodgrass, M. Mateas, and S. Ontañón, "The VGLC: The video game level corpus," in *PCG Workshop at 1st Joint International Conference of DiGRA and FDG*, 2016.

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

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

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

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] G. J. Székely and M. L. Rizzo, "Energy statistics: A class of statistics based on distances," *Journal of statistical planning and inference*, vol. 143, no. 8, pp. 1249–1272, 2013.

[34] A. Summerville, "Expanding expressive range: Evaluation methodologies for procedural content generation," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[35] S. Cooper and A. Sarkar, "Pathfinding agents for platformer level repair," in *EXAG Workshop*, 2020.