# Latent Combinational Game Design

Anurag Sarkar, Seth Cooper

*Abstract*—We present *latent combinational game design*—an approach for generating playable games that blend a given set of games in a desired combination using deep generative latent variable models. We use Gaussian Mixture Variational Autoencoders (GMVAEs) which model the VAE latent space via a mixture of Gaussian components. Through supervised training, each component encodes levels from one game and lets us define blended games as linear combinations of these components. This enables generating new games that blend the input games as well as controlling the relative proportions of each game in the blend. We also extend prior blending work using conditional VAEs and compare against the GMVAE and additionally introduce a hybrid conditional GMVAE (CGMVAE) architecture which lets us generate whole blended levels and layouts. Results show that these approaches can generate playable games that blend the input games in specified combinations. We use both platformers and dungeon-based games to demonstrate our results.

*Index Terms*—procedural content generation, combinational creativity, game blending, variational autoencoder

## I. Introduction

Methods for Procedural Content Generation via Machine Learning (PCGML) [1] primarily focus on learning distributions of individual games, but an emerging body of work [2] has focused on methods for recombining learned models and distributions across multiple games to generate content for entirely new games. These have been motivated by limited training data, wanting to leverage design knowledge in one set of games to apply in another and exploring creative PCGML applications beyond level generation. One such application is game blending—the generation of new games by blending the levels and/or mechanics of existing games, inspired by designers building new games by combining ideas from existing ones [3]. An extensively used approach for blending levels has been the variational autoencoder (VAE) [4], an encoder-decoder architecture that learns latent representations of data. Prior works [5], [6] have demonstrated VAEs to be capable of such blending but offer limited means to control the blending, learning a combined latent space spanning all games without affording the ability to sample specific blends or games. Given a design space spanning a set of games, one may wish to blend only a certain subset of the games and in desired proportions relative to one another. To this end, prior works [7], [8] have shown via conditional VAEs that supervision via labels is promising for obtaining such desired blend combinations. However, these works only examined binary combinations (a game is either included or not) rather than relative blend ratios. Further, the playability of such blended levels has been previously evaluated only by using mechanics for the original games as proxies, rather than by blending the mechanics themselves.

A. Sarkar and S. Cooper are both with the Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA (email: sarkar.an@northeastern.edu, se.cooper@northeastern.edu)

In this work, we introduce a framework for generating playable blended games that blend desired proportions of games. We call this *latent combinational game design—latent* since we use latent representations, *combinational* since game blending is a combinational creativity [9] process and *game design* since we generate novel, playable games. Overall, the framework consists of:

- Training a latent variable model to learn a design space that captures the set of $k$ games to be blended
- Defining $k$ weights specifying blend proportions
- Using the weights to generate blended levels by sampling from the relevant parts of the learned design space
- Using the weights to derive a blended agent that combines the mechanics of the original game agents

We apply this framework using supervised Gaussian Mixture VAEs (GMVAE) [10], [11] and conditional VAEs (CVAE) [12] which we train using levels from the platformers *Super Mario Bros.*, *Kid Icarus*, *Mega Man* and *Metroid*. Moreover, unlike prior approaches, we test playability using blended agents derived by combining the jump physics of these games using the same weights used to blend the levels. Overall, we find that the framework (with both models) enables generating playable blended games based on the weights. Further, since these models generate segments, we introduce a novel hybrid conditional GMVAE (CGMVAE) architecture which enables generating whole blended levels as well as layouts that blend levels of the dungeon games *The Legend of Zelda*, *DungeonGrams* and a repurposed version of *Metroid*, thereby generalizing to beyond platformers. This work thus contributes:

- a controllable combinational creativity framework for blending games in terms of both levels and mechanics
- implementations of the framework using both GMVAEs and CVAEs
- to our knowledge, the first use of blended jump agents
- a novel CGMVAE model combining GMVAEs and CVAEs

## II. Related Work

While PCGML [1] methods typically focus on modeling individual games, a body of work has emerged which instead recombines models from different games to generate new forms of content and explore design spaces across multiple games. Such works have recently been coined as methods for PCG via Knowledge Transformation [2] and include game generation by recombining learned game graphs [13] and learning mappings between different platformers [14], to name a few. A specific focus has been game blending i.e., generating new games by blending the levels and/or mechanics of two or more games, introduced in [3] where specifications of *Frogger* and *Zelda* were blended manually. Several works have since implemented blending using variational autoencoders (VAEs) [4] which learn continuous latent representations that
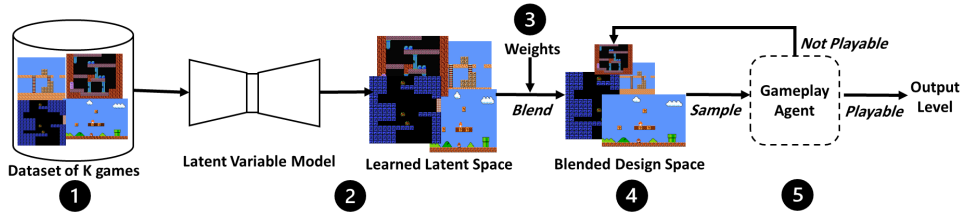
Fig. 1: *Latent combinational game design:* 1) a dataset of k games for training 2) a model which learns a latent space on which 3) blend weights are applied to yield 4) a blended design space for sampling content satisfying 5) a constraints module such as a gameplay agent.

span all input games and can be sampled and searched for blended content, having been used for blending platformers [5], [6] as well as platformers with dungeons [8]. These works however learn a fixed blend space without readily allowing the specification of different blend combinations such as blending a subset of the input games and in specific proportions. In this work, we implement controllable blending using Gaussian Mixture (GMVAE) and conditional VAEs (CVAE). GMVAEs [10], [11] use a mixture of Gaussians as the prior distribution for the latent space unlike regular VAEs which use a standard Gaussian. This enables GMVAEs to cluster data in an unsupervised manner, where each GM component encodes a specific cluster and can be used to generate data belonging to it. Previously, GMVAEs have been used to discover clusters of platformer levels [15]. We instead use supervised GMVAEs since we want each GM component to cluster levels of one of the input games and we know which game each level belongs to. To our knowledge, this is the first use of a supervised GMVAE in a games context, though prior works have applied supervised [16] and semi-supervised GMVAEs [17] outside of games. CVAEs [12], [18] add supervision to regular VAEs through labeling. Prior works [7], [8] have used CVAEs to control blending using multi-hot labels that specify which games to include in the blend. We additionally apply continuous labels to control the relative ratios of each game.

Game blending is a conceptual blending [19] method, falling under combinational creativity [9], the branch of creativity concerned with generating new concepts by recombining existing ones. A prior method for generating new games similar to our approach is conceptual expansion [13]. Our works differ in that their system does automated game recombination and uses game graphs and video input whereas we generate specific combinations of games and use VAE latent representations and text-based input. In being a creative ML-based method for game design, our work can also be viewed as a GDCML (Game Design via Creative Machine Learning) approach [20].

Few previous works have attempted blending gameplay. Prior work [21] has demonstrated the use of a constraint-based method for blending the reachability rules of different games but the only prior instance of blending jumps can be found in [22] where blended jumps are extracted from paths in generated blended levels. We blend jumps of the original games directly using jump models from [23] which learned hybrid automata to describe jumps of NES-era platformers.

## III. FRAMEWORK

We discuss our overall methodology in two sections. In this section, we introduce the framework and discuss its

implementation via GMVAEs and CVAEs and its use for platformer level and jump blending. In the section after, we will discuss the CGMVAE and CCVAE architectures and how we use them to generate whole blended levels and extend to dungeons. The framework (Figure 1) consists of:

1) *Data* comprised of levels from $k$ different games
2) A *model* that learns a representation that enables working with different combinations of the $k$ games
3) A set of $k$ *weights* to specify how each game should be combined in the final blend
4) A *blended design space* obtained by blending the $k$ games using the weights
5) A *constraints module* that checks if outputs satisfy constraints such as playability.

A dataset of $k$ games defines the full blending possibility space and is used to train a latent variable model capable of learning a disentangled[1] representation of the games. We use $k$-component GMVAEs and CVAEs with $k$-dimensional labels which allow manipulating each game via a separate GM component and label dimension respectively (note that this could be any model that allows reasoning about the $k$ subsets of data separately, such as a conditional GAN [24]). To blend the games, we use $k$ weights defining the desired proportion of each game. Applying these gives us the final blended space with the $k$ games blended in the specified proportions. For the GMVAE, we linearly combine the $k$ GM components to obtain a new blended distribution. For the CVAE, we use the $k$ weights as the $k$-dimensional label when generating a level. The final blended design space is then sampled to produce the blended outputs. Lastly, a module checks if the sampled output satisfies certain constraints. Here, the constraint is that outputs must be playable by blended agents, however this is a general module not restricted to playtesting e.g., it could be a module that uses evolutionary algorithms to search the design space for specific content. Borrowing linear algebra vocabulary, we can consider the original $k$ games as the *basis games* for blending and each blended game is a *linear combination* of these basis games, specified by the blend weights. The set of all such linear combinations is the *span* of these basis games.

### A. Level Data

Training data consisted of levels of *Super Mario Bros. (SMB)*, *Kid Icarus (KI)*, *Mega Man (MM)* and *Metroid (Met)*, taken from the Video Game Level Corpus (VGLC) [25]. VGLC levels use a text format with unique characters mapping

---

[1]We use *disentangled* in the literal sense (i.e., the model learns separate representations for each game) and not to refer to the specific technique of making latent dimensions learn independent factors of variation.
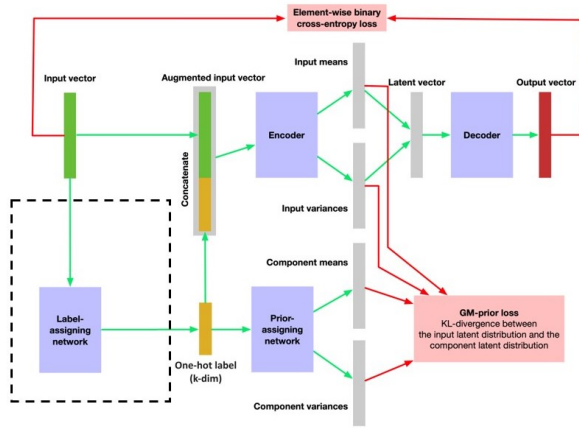
Fig. 2: GMVAE architecture. The label-assigning network (dotted box) is part of the unsupervised GMVAE (as in prior work [15]) but not required in the supervised GMVAE setting used in this work.

to specific tiles. To show blending, we used a distinct mapping for each game, modifying the original tiles when games shared the same tile characters (e.g., enemies in both SMB and Met use 'E'). We extracted 15x16 non-overlapping segments from each level based on the dimensions of playable areas in MM and Met where horizontal and vertical sections are 15 and 16 tiles high and wide respectively. KI levels are also 16 tiles wide. The 14-tile high SMB levels were padded with a row of background tiles on top. For Met, we filtered out entirely solid segments. We extracted 172, 80, 143 and 435 segments for SMB, KI, MM and Met respectively and upsampled the non-Met segments to obtain a total of 1740 segments. Each segment had a length-4 one-hot label indicating which game it belonged to, with $\langle 1000 \rangle$, $\langle 0100 \rangle$, $\langle 0010 \rangle$ and $\langle 0001 \rangle$ indicating SMB, KI, MM and Met respectively.

### B. Gaussian Mixture VAEs

VAEs [4] are deep latent variable models consisting of encoder and decoder networks. For a $z$-dimensional latent space, for each input training instance, the encoder outputs $z$ pairs of means and variances which parameterize the latent distribution. A $z$-dimensional latent vector sampled from this distribution is then forwarded through the decoder to obtain outputs. Training is done by minimizing a loss function composed of—1) the reconstruction error between encoder inputs and decoder outputs and 2) the KL-divergence between the latent distribution and a tractable prior (typically standard Gaussian). Rather than use a standard Gaussian as the prior for the latent distribution, GMVAEs [10], [11] use a mixture of Gaussians. For a mixture of $k$ Gaussians, this is done via: 1) using a $k$-dimension one-hot label indicating the component for an input, 2) training an additional network that maps a one-hot label to pairs of means and variances that parameterize the 1 out of $k$ components the input belongs to and 3) modifying the 2nd loss term above to instead compute KL-divergence between the latent and component distributions. GMVAEs are typically unsupervised with a label-assigning network used to learn the one-hot labels. This was used in [15] for discovering clusters of platformer levels. For blending however, we want each component to encode the segments of a specific game

and since we know which game each segment belongs to, we use a supervised GMVAE, manually supplying one-hot labels for each segment, making the label-assigning network unnecessary and simplifying training, as shown in Figure 2. Thus, a $k$-component supervised GMVAE trained on levels from $k$ games results in each component encoding the levels for one game. Each game is thus modeled by a separate Gaussian distribution parametrized by a mean and a variance learned through training. For generating a blended game, we accept $k$ weights and model the blended game as a new distribution defined by the linear combination of the $k$ GMs, as specified by the weights. For $k = 4$ as in this work, after training we obtain 4 GM components with means and variances given by: $M = [\mu_1, \mu_2, \mu_3, \mu_4]$ and $V = [\sigma_1^2, \sigma_2^2, \sigma_3^2, \sigma_4^2]$. We then accept a weight vector $W = [w_1, w_2, w_3, w_4]$ indicating the weight of each GM desired in the new blended game. We then define a new blended game distribution as the linear combination of the 4 GMs with a mean given by the inner product $\langle M, W \rangle$ and variance given by the inner product $\langle V, W^2 \rangle$. This new distribution is the blended design space from the framework and is sampled to generate levels for the new blended game.

### C. Conditional VAEs

Another approach for blending different combinations of games explored in past works involves conditional VAEs (CVAE) which are supervised VAEs that use labeled inputs. The encoder and decoder learn to use labels to encode inputs and produce outputs respectively. New outputs are generated by sampling a latent vector, concatenating the desired label and forwarding it through the decoder. Blending games using CVAEs involves using one-hot labels for training but multi-hot labels for generation e.g., $\langle 1011 \rangle$ indicates a blend of SMB/MM/Met without KI. Prior works have used CVAEs to blend games using multi-hot labels despite being trained only using one-hot labels. However, these labels only allow a game to be included or excluded from the blend. Since our goal is to generate arbitrary blends of games, we test if CVAEs trained using the same one-hot labels can work with arbitrary labels i.e., if similar blend weights used to linearly combine the learned components of the GMVAE, could be used by the CVAE to produce blended games that combine the games in accordance with the weights. After training the CVAE on the same data and labels as the GMVAE, we generate blended levels by: 1) sampling vectors from the CVAE latent space 2) accepting $k$ blend weights as before but using them as the $k$-element label 3) concatenating it to each sampled vector and 4) forwarding the concatenated vectors through the decoder to obtain the output blended levels.

### D. Blending Jumps

Prior works [6], [26] have evaluated playability of blended platformer levels using separate A* agents for each game. By seeing which game's agent can play through a blended level, we can conclude if a blended level is e.g., more Mario-like or Metroid-like. However, since our goal is generating new blended games, we blend gameplay in addition to blending levels. We define a blended jump model as a linear combination of the parameters of the original jump models, using

the same weights used to blend levels. Thus, the jump for a blended game is a linear combination of the jumps of the original games. For jump models, we use the models from prior work [23] that learned hybrid automata for describing the jump physics of several NES platformers, including the four we use. The automata consist of parameters defining the jump arcs and vary depending on the game. We obtain blended jump models by linearly combining these parameters using the blend weights. The automata can be used to derive jump arcs for the respective games. These jumps, when represented as lists of (x,y) coordinates, can in turn be used to learn the impulse and gravity parameters that define the jump, as described in [22]. This has two specific benefits—1) jump arcs derived via blending can be used with the Summerville tile-based A* agent [27] to quickly test if they can be used to traverse blended levels and 2) in the future, the impulse and gravity parameters could be used to define the mechanics for a player controller so that the blended levels can be played in an interactive environment.

### E. Experiments

All models were implemented using PyTorch [28]. The GMVAE consists of an encoder, decoder and prior-assigning network. The encoder and decoder consisted of 4 and 3 fully-connected layers respectively, with both using ReLU activation. The last encoder layer was split into 2 parts, each having 1 fully-connected layer with the latter additionally using softplus activation. These two layers output the means and variances of the latent distribution respectively. The prior assigning network consisted of 2 independent sub-networks each having 1 fully-connected layer with the second additionally using Softplus activation. The two networks output the component means and variances respectively. Training was done for 1000 epochs using the Adam optimizer with an initial learning rate of 0.001, decayed by 0.1 every time training plateaued for 50 epochs. For the CVAE, the encoder and decoder consisted of 4 and 3 fully-connected layers respectively, both using ReLU activation. Training lasted 10000 epochs using Adam with an initial learning rate of 0.001, set to decay by 0.1 every 2500 epochs. The weight on the KL term of the loss function was annealed from 0 to 1 for the first 2500 epochs. Hyperparameters were determined based on prior works using these models and through experimentation. For evaluation, we conducted a 3-part study measuring 1) how accurately the output of the blend models adhered to the blend weights, 2) playability of the blended levels using the blended agents and 3) comparing the tile patterns and content in the generated blended levels with those in the original levels. Additionally, for all evaluations, we used two types of blend weights—1) binary weights in the form of length-4 one-hot and multi-hot vectors indicating single-game and multi-game blends respectively and 2) fractional weights consisting of length-4 floating-point vectors, representing any arbitrary combination of games. For binary weights, we evaluate all possible length-4 binary vectors except $\langle 0000 \rangle$ since it is unclear how to reason about a blend containing none of the games, giving us $2^4 - 1 = 15$ vectors. For fractional weights, we define 4 vectors to test against to cover a range of blend

| | Binary | | | | Fractional | | | |
|---|---|---|---|---|---|---|---|---|
| | 8-dim | 16-dim | 32-dim | 64-dim | 8-dim | 16-dim | 32-dim | 64-dim |
| CVAE | 1021.99 | 741.14 | 939.04 | **700.14** | 1113.8 | 995.08 | 1171.03 | **939.46** |
| GM | 659.83 | 707.9 | **293.79** | 310.94 | 904.21 | 937.26 | **851.84** | 947.31 |

TABLE I: Classification results for blended levels generated using binary and fractional weights. Lower scores are better.

combinations: $\langle 0.5, 0.3, 0.2, 0.0 \rangle$, $\langle 0.1, 0.1, 0.1, 0.7 \rangle$, $\langle 0.1, 0.6, 0.2, 0.1 \rangle$ and $\langle 0.0, 0.2, 0.3, 0.5 \rangle$. For each of the CVAE and GMVAE, we tested 4 different latent dimensionalities of 8, 16, 32 and 64 but found none to be clearly preferable across all experiments, with different sizes producing best results for different metrics. For clarity and space, instead of presenting results for all 8 models, we present results from the GMVAE-32 and CVAE-64 models as they performed the best on the blend accuracy evaluation which is the most relevant for the main focus of this work i.e., controllable blending of games.

*1) Blend Accuracy:* For evaluating how well the outputs of the blended models adhere to the blend weights, we used scikit-learn [29] to train a random forest classifier on the segments using the game of that segment as the class label. We achieved a 98.19% accuracy on an 80-20 train-test split. The parameters for the classifier were determined via grid search. This classifier approach for evaluating blending has been used in prior CVAE works and acts as a proxy since we lack ground-truth blended levels to compare generated output. The expectation is that when using single game weights (e.g., $\langle 0,0,0,1 \rangle$, $\langle 1,0,0,0 \rangle$), classifier predictions for a game will be high when its bit is set to 1 and low when set to 0. Similarly, when using blended weights (e.g., $\langle 1,1,0,0 \rangle$, $\langle 0,1,0,1 \rangle$), the predictions should be more spread out across the games whose bit is set to 1. When using fractional weights (e.g., $\langle 0.5, 0.2, 0.3, 0.0 \rangle$), the predictions should be spread out in accordance with these weights. For classifying binary weights, for each model, we sampled 1000 latent vectors for each possible 4-digit binary weight, leaving out $\langle 0,0,0,0 \rangle$. We applied the classifier on each generated segment and tracked the percentage of times each of the 4 games was predicted, per 4-digit weight. To determine how well the outputs matched a given weight, for each weight, we computed a score based on the following formula: $S = \sum_{i=1}^{4}(w_i * f - p_i)^2$ where $w_i$ is the $i$th weight element indicating the desired weight of the $i$th game, $f$ is a factor equal to 100 divided by the number of ones in the weight and $p_i$ is the percentage of the 1000 segments for which the classifier predicted the $i$th game. The lower the score, the better the model matched outputs with the given weights with 0 representing a perfect match, e.g., if the weights are $\langle 1,0,1,0 \rangle$, the factor $f$ is 100/2=50 and a perfect score of 0 is achieved when the classifier predicts games 1 and 3 50% of the time each. For fractional weights, we generated 1000 latents per weight per model and computed scores similarly, except here $f$ is set to a constant 100. Results are given in Table I. For both binary and fractional weights, the GMVAE does better than the CVAE for all latent dimensions except for 64 where the CVAE does better. Looking at the latent sizes, for the GMVAE and CVAE, 32 and 64 dimensions respectively give the lowest score when combining binary and fractional weights and thus for the remainder of our evaluations, we will only

| Blend Weights | GM-32 dim | | | | | CVAE-64 dim | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SMB | KI | MM | Met | Score | SMB | KI | MM | Met | Score |
| ⟨0001⟩ | 0.7 | 0.1 | 0.4 | 98.8 | 2.1 | 0.1 | 0 | 0 | 99.9 | 0.02 |
| ⟨0010⟩ | 5.2 | 1.8 | 93.0 | 0 | 79.3 | 1.5 | 0.9 | 97.6 | 0 | 8.8 |
| ⟨0011⟩ | 4.4 | 1.3 | 40.5 | 53.8 | 125.7 | 6.2 | 0 | 58.4 | 35.4 | 322.2 |
| ⟨0100⟩ | 2.3 | 96.5 | 1 | 0.2 | 18.6 | 0.4 | 99.6 | 0 | 0 | 0.3 |
| ⟨0101⟩ | 11.7 | 21.1 | 2.1 | 65.1 | 1204.5 | 17.7 | 5.5 | 0.6 | 76.2 | 2980.3 |
| ⟨0110⟩ | 11.8 | 38 | 50 | 0.2 | 283.3 | 23.8 | 45.7 | 30.5 | 0 | 965.2 |
| ⟨0111⟩ | 15.6 | 20.4 | 34.1 | 29.9 | 422 | 35.5 | 7.5 | 30.9 | 26.1 | 1983.5 |
| ⟨1000⟩ | 94.6 | 3.8 | 1.6 | 0 | 46.2 | 98.6 | 1.4 | 0 | 0 | 3.9 |
| ⟨1001⟩ | 34.8 | 2.1 | 2.3 | 60.8 | 357.4 | 39.9 | 1.4 | 0 | 58.7 | 179.7 |
| ⟨1010⟩ | 48.1 | 5.3 | 46.6 | 0 | 43.3 | 42.7 | 9.3 | 48 | 0 | 143.8 |
| ⟨1011⟩ | 39 | 2.8 | 38.8 | 19.4 | 263.8 | 40.8 | 6.1 | 36.9 | 16.2 | 398.8 |
| ⟨1100⟩ | 56 | 42.8 | 1.2 | 0 | 89.3 | 64.5 | 35.5 | 0 | 0 | 420.5 |
| ⟨1101⟩ | 50.6 | 25.7 | 3 | 20.7 | 524.8 | 55.9 | 22.1 | 0.2 | 21.8 | 768.5 |
| ⟨1110⟩ | 47.8 | 26.4 | 25.7 | 0.1 | 315.6 | 30.2 | 62.6 | 7.2 | 0 | 1549.3 |
| ⟨1111⟩ | 42.5 | 19.4 | 29.6 | 8.5 | 631 | 40.5 | 37.1 | 13.4 | 9 | 777.2 |
| ⟨0.5, 0.3, 0.2, 0⟩ | 74.4 | 18.6 | 7 | 0 | 894.3 | 65.9 | 33.1 | 1 | 0 | 623.4 |
| ⟨0.1, 0.1, 0.1, 0.7⟩ | 2.3 | 0.7 | 1.8 | 95.2 | 848.1 | 2.4 | 0.2 | 0.5 | 96.9 | 967.7 |
| ⟨0.1, 0.6, 0.2, 0.1⟩ | 12.6 | 77.3 | 9.3 | 0.8 | 505.2 | 9.8 | 88.9 | 1.2 | 0.1 | 1286.7 |
| ⟨0, 0.2, 0.3, 0.5⟩ | 8.3 | 3.1 | 14.7 | 73.9 | 1159.8 | 12.3 | 2.5 | 18.3 | 66.9 | 880 |

TABLE II: Full classification results for both blend weight types. Lower the score, more aligned are predictions with weights.
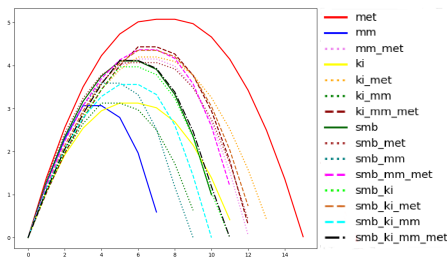


Fig. 3: Jump arcs for various combinations of games.

present and discuss results obtained using the 32-dimensional GMVAE and 64-dimensional CVAE. Full classification results are shown in Table II. We see that in most cases, the game predicted most often is one whose bit is set to 1 in the binary case. When using fractional weights, the amounts that levels are classified as different games corresponds to the value of the weights, though very loosely. Interestingly, the CVAE is able to reason with fractional weights despite having only been trained using one-hot binary weights as labels. Overall, this suggests that both GMVAE and CVAE models are able to generate output that corresponds to the desired blend weights. These results give some intuition about why the GMVAE outperforms the CVAE in Table I, namely that the GMVAE is better able to blend the individual games in accordance with the weights while the CVAE is more prone to pushing the blend towards the game(s) with highest weight, as shown (in Table II), by how for most multi-game blends, the CVAE scores higher (worse) than the GMVAE, usually due to putting too much weight on one of the games whose bit is set to 1. This may be due to how these models perform blending. GMVAEs learn separate distributions per game which are then combined using the weights where as CVAEs learn one distribution spanning all games and blends are produced by applying labels (i.e., weights) to samples from this distribution. If this single distribution approximates some games better than others, the labels may be insufficient in producing accurate blends.

*2) Playability:* We evaluated playability of generated blended levels using jump arcs obtained by blending the automata models using given blend weights and the Summerville tile-based A* agent [30]. The original VGLC version worked with solid and non-solid tiles and was updated in [22] to work with 4 affordances [31]: solids, hazards, passables and climbables. We further updated the agent to find start and goal tiles in a segment. A segment is then playable if the agent can find a path from start to goal. If no path or goal is found, it is unplayable. The jumps used by the blend agent are derived from blended automata models which in turn are obtained by blending the hybrid automata models of the original 4 games using the blend weights. Details about these automata and jump extraction can be found in [23]. Blended jump arcs are shown in Figure 3. Since the blend agent simulates an agent that can play blends of all 4 games, it knows the tile-to-affordance mappings of all 4 games. For each model, we sampled 1000 segments for each possible 4-digit binary weight and computed the percentage that were playable by the blended agent obtained by blending the original jump models according to the corresponding weight. Since games can progress both horizontally and vertically, for each segment, the agent looks for both a valid horizontal and vertical path. If either one is found, the segment is playable. To simplify computations, we only look for left-to-right and bottom-to-top paths. Thus, our evaluations underestimate playability since e.g., MM and Met segments that progress downward but do not have an upward moving path would be deemed unplayable. Full playability results for the blended agent on binary and fractional-weighted blends are shown in Tables III and IV. Table IV also shows how well the original agents did in the blended levels to depict that performance roughly corresponds to the game's weight in the blend e.g., for ⟨0.5, 0.3, 0.2, 0⟩, the SMB agent completes more levels than the KI agent which completes more than the MM agent which does better than the Met agent. We show the performance of the original agents to show that all models generate levels in accordance with the weights and not to compare playability of the blend agent with the original agents since such comparisons are not useful. Essentially, there are two broad ways to compare. First, we can assume that the original agent for a game should not reason about the tiles of another e.g., an SMB agent should not know to avoid MM spikes. But if only the blend agent knows all tiles across all games, then it will trivially do the best since in most blended levels, the original agents would face tiles that are undefined for them and they would not be able to find paths. Thus, playability reduces to tile knowledge. Second, we can instead assume that all agents work at the affordance level, agnostic of game-specific tiles. Here, playability is completely determined by an agent's jumpsize. Higher the jump, higher the playability when all agents work with the same tile information. Since the blend agent combines the jumps of all games, its jump by definition can never be bigger than that of Metroid. Thus, playability reduces to jumpsize. Hence, in either case, such comparisons are uninformative. Overall, we observe satisfactory playability of 61.97% and 65.58% for the GMVAE, averaged across binary and fractional weights respectively and 70.14% and 68.03% for the CVAE.

*3) Tile Metrics:* To compare the content of generated blended levels with those in the original levels, we used the Tile Pattern KL-Divergence (TPKLDiv) [32] metric which measures the similarity between two sets of levels in terms

| ⟨SMB, KI, MM, Met⟩ | GM-32 dim | CVAE-64 dim |
|---|---|---|
| ⟨0,0,0,1⟩ | 67.4 | 68.2 |
| ⟨0,0,1,0⟩ | 49 | 48.3 |
| ⟨0,0,1,1⟩ | 62.3 | 63.5 |
| ⟨0,1,0,0⟩ | 69.3 | 68.9 |
| ⟨0,1,0,1⟩ | 52.2 | 70.1 |
| ⟨0,1,1,0⟩ | 52.7 | 72.2 |
| ⟨0,1,1,1⟩ | 53.5 | 76.9 |
| ⟨1,0,0,0⟩ | 83.8 | 89.8 |
| ⟨1,0,0,1⟩ | 69.1 | 71.1 |
| ⟨1,0,1,0⟩ | 70.5 | 70.5 |
| ⟨1,0,1,1⟩ | 74.9 | 73.9 |
| ⟨1,1,0,0⟩ | 82.3 | 84.2 |
| ⟨1,1,0,1⟩ | 73.9 | 68.5 |
| ⟨1,1,1,0⟩ | 68.1 | 61.5 |
| ⟨1,1,1,1⟩ | 69.9 | 64.5 |

TABLE III: Percentage of playable levels for each binary blend.

| | GM-32 dim | | | | | CVAE-64 dim | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Blend | SMB | KI | MM | Met | Blend | SMB | KI | MM | Met |
| ⟨0.5, 0.3, 0.2, 0⟩ | 75.3 | 67.3 | 8.5 | 6.5 | 0.2 | 77.9 | 66.4 | 21.8 | 4.6 | 2.2 |
| ⟨0.1, 0.1, 0.1, 0.7⟩ | 62.1 | 0.2 | 0.6 | 0.7 | 66.4 | 61.8 | 0.5 | 0.3 | 1.2 | 64.8 |
| ⟨0.1, 0.6, 0.2, 0.1⟩ | 71.3 | 9.9 | 58 | 6.9 | 1.5 | 71.9 | 3.1 | 63.2 | 8.2 | 3 |
| ⟨0, 0.2, 0.3, 0.5⟩ | 53.6 | 0.7 | 3.7 | 9.7 | 51.3 | 60.5 | 0.6 | 2.5 | 14.7 | 58.9 |

TABLE IV: Percentage of playable levels for fractional blends using the blended agent and original agents for each game.

| | GM-32 dim | | | | CVAE-64 dim | | | |
|---|---|---|---|---|---|---|---|---|
| Blend Weights | SMB | KI | MM | Met | SMB | KI | MM | Met |
| ⟨0001⟩ | 12.68 | 13.02 | 12.83 | 1.59 | 13.51 | 12.97 | 13.16 | 1.37 |
| ⟨0010⟩ | 10.95 | 10.64 | 2.1 | 11.28 | 9.87 | 10.23 | 1.64 | 11.03 |
| ⟨0011⟩ | 11.28 | 10.69 | 7.59 | 5.59 | 8.7 | 8.78 | 6.37 | 5.2 |
| ⟨0100⟩ | 10.77 | 2.6 | 10.55 | 11.7 | 11.02 | 1.62 | 10.97 | 11.61 |
| ⟨0101⟩ | 10.84 | 8.57 | 11.07 | 4.46 | 9.04 | 8.16 | 9.16 | 2.96 |
| ⟨0110⟩ | 8.58 | 6.74 | 5.34 | 10.03 | 7.95 | 4.79 | 5.14 | 8.52 |
| ⟨0111⟩ | 9.01 | 7.61 | 5.78 | 6.35 | 6.74 | 6.65 | 5.15 | 4.26 |
| ⟨1000⟩ | 0.8 | 4.67 | 4.31 | 4.49 | 0.65 | 4.43 | 4.26 | 4.64 |
| ⟨1001⟩ | 6.76 | 7.73 | 7.49 | 2.36 | 5.03 | 5.84 | 5.78 | 1.9 |
| ⟨1010⟩ | 3.53 | 6.15 | 2.51 | 6.4 | 2.94 | 4.09 | 1.81 | 4.49 |
| ⟨1011⟩ | 5.31 | 6.29 | 4.25 | 4.42 | 3.95 | 4.56 | 3.18 | 3.22 |
| ⟨1100⟩ | 3.6 | 4.25 | 5.97 | 6.51 | 2.44 | 2.48 | 4.12 | 4.06 |
| ⟨1101⟩ | 4.78 | 4.06 | 6.1 | 4.32 | 2.58 | 3.03 | 3.13 | 1.49 |
| ⟨1110⟩ | 4 | 5.29 | 3.94 | 6.56 | 2.19 | 2.02 | 2.46 | 3.54 |
| ⟨1111⟩ | 4.37 | 5.66 | 4.08 | 5.36 | 3.26 | 3.85 | 2.76 | 2.3 |
| ⟨0.5, 0.3, 0.2, 0⟩ | 2.01 | 5 | 4.5 | 5.38 | 2.38 | 3.5 | 4.34 | 4.81 |
| ⟨0.1, 0.1, 0.1, 0.7⟩ | 11.77 | 12.04 | 11.66 | 1.63 | 11.4 | 11.05 | 11.15 | 1.46 |
| ⟨0.1, 0.6, 0.2, 0.1⟩ | 8.7 | 3.32 | 8.24 | 9.54 | 8.26 | 2.11 | 8.19 | 8.87 |
| ⟨0, 0.2, 0.3, 0.5⟩ | 11.19 | 10.5 | 10.01 | 3.79 | 9.48 | 9.28 | 8.26 | 3.64 |

TABLE V: TPKLDiv values between original games and the different blend configurations. Lower the value, closer the tile-pattern distribution of that blend is to the corresponding game.
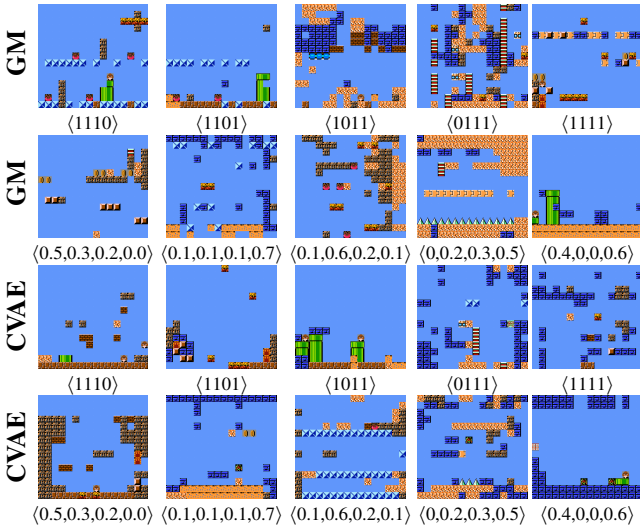


Fig. 4: Example generated levels using the GM and CVAE models for a number of different blend weights.

of the KL-divergence between their tile pattern distributions. For each model and set of blend weights, we generate 1000 levels and compare the average TPKLDiv between them and each set of original levels, separately for each of the 4 games, with the values averaged over 2x2, 3x3 and 4x4 patterns. We expect lower values when comparing the original levels of a game with blended levels produced by weights that include that game, and higher values when they do not, e.g., using ⟨1,1,0,0⟩ should lead to lower TPKLDiv values compared to SMB and KI than with MM and Met. Full results are in Table V and are true to this expectation, giving further support that the models can produce blends in accordance with the weights.

### F. Visual Inspection

Figure 4 shows example blended segments. Note that a sampled segment may not appear blended since weights apply to whole distributions and not individual segments. Thus it is possible to sample segments that contain content from only 1

game, highlighting the need to use more sophisticated methods beyond just random sampling. That said, these examples depict the games being blended. SMB can be identified by its pipes and goombas, KI by its blue platforms, red hazards, brown tiles and doors, MM via orange tiles, spikes and ladders and Metroid via dark blue tiles, orange lava and metroid creatures.

## IV. GENERATING AND BLENDING WHOLE LEVELS

In this section, we describe our approach for extending the LCGD framework in 2 ways—1) generating whole levels and 2) blending levels from dungeon-based games. For this, we combine the GMVAE and CVAE to introduce a novel Conditional Gaussian Mixture VAE (CGMVAE) architecture which uses a supervised GM prior for the VAE latent space thus still enabling games to be encoded as GM components but now, we also condition the encoder/decoder with labels indicating segment orientation i.e., the direction(s) in which movement is possible across the segment. Directional labeling was introduced in prior work [8] for generating whole levels. Using labels for progression enables generating segments that have desired orientations and can be reliably connected together to form whole levels. For dungeons, such labels can control the direction(s) in which rooms have doors/openings and enable generating dungeons with appropriately interconnected rooms. This in turn lets us use the CGMVAE to apply the LCGD framework on dungeon-based games. We compare this with a CVAE where a label controlling direction is concatenated to the label controlling blending. To distinguish it from the prior CVAEs, we refer to it as CCVAE to indicate that the labels here control both blend ratio and directionality.

### A. Model and Architecture

The architectures and training for both CGMVAE and CC-VAE were identical to the GMVAE and CVAE but augmented by the length-4 multi-hot labels. For both models, these labels were concatenated to both the encoder and decoder inputs. The CGMVAE architecture is shown in Figure 5. The reasoning for this labeling is that whole levels can be considered to be composed of segments that are connected together so as
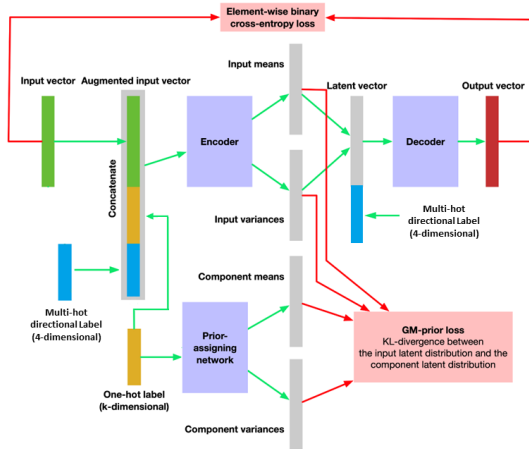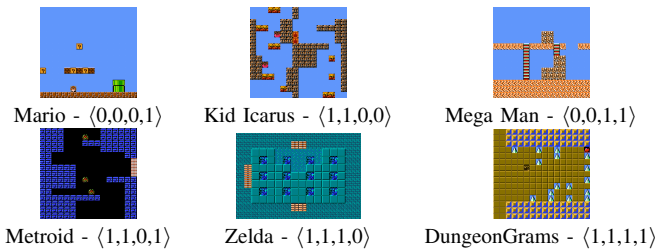
Fig. 5: Conditional GMVAE (CGMVAE) architecture.



Mario - $\langle 0,0,0,1 \rangle$ · Kid Icarus - $\langle 1,1,0,0 \rangle$ · Mega Man - $\langle 0,0,1,1 \rangle$

Metroid - $\langle 1,1,0,1 \rangle$ · Zelda - $\langle 1,1,1,0 \rangle$ · DungeonGrams - $\langle 1,1,1,1 \rangle$

TABLE VI: Example segments with corresponding directional labels indicating doors/openings in $\langle$Up, Down, Left, Right$\rangle$ directions.

to enable progressing through them. Platformer and dungeon levels can be viewed as made up of discrete segments and rooms respectively. For the CCVAE, we concatenated the new directional label to the game label used with the CVAE models.

## B. Level Data

The training data included levels from both platformers and dungeon-based games. The platformer models were trained using the same data as before, with directional labels added to the game labels. Directional labels were length-4 multi-hot vectors with elements corresponding to up, down, left and right respectively and set to 1/0 to indicate if progression in the corresponding direction was/was not possible e.g., $\langle 1,1,0,0 \rangle$ indicates a segment where progress can be made upward or downward but not left or right. The label for each segment was determined manually via visual inspection and indicated the directions in which a segment could be entered or exited by a player. For Metroid segments, directions having a door were considered open. A challenge for training on dungeon-based games is the lack of data in the VGLC where the only such game with text-based level data is *The Legend of Zelda*. Thus, to blend multiple such games, we also used levels from *DungeonGrams* (DGG), a roguelike dungeon crawler developed for prior research [33]. We also leveraged the previously used Metroid levels. While Metroid is a platformer, its sprawling, interconnected world can be seen as a dungeon from a top-down perspective. We trained on 15x16 segments from each game. Since Zelda and DGG levels are 11x16, we padded each by duplicating the outermost rows. We also added vertically and horizontally flipped versions of each Zelda

room, if not already present. We obtained 502 Zelda rooms, 522 DGG levels and 435 Metroid segments but upsampled Zelda and Metroid to get a total of 1566 segments. Example segments and directional labels are shown in Table VI.

## C. Layout Algorithm

To arrange sampled segments/rooms into whole levels, we use the layout algorithm from [8]. This starts with an initial segment location closed in all four directions. In each iteration, we randomly select a closed side of the current location and place a new location next to that side if there is none. The closed side just selected is then labeled open to connect the new location to the prior one. This is repeated until a desired number of interconnected locations are generated. Generating a platformer layout is simpler since movement is overall left-to-right, with/without vertical sections in between. For this, we randomly set an upward or rightward direction for the initial segment and set directions for subsequent segments such that progress is possible from start to finish (e.g., segment following a rightward/upward segment should be open on the left/bottom respectively). There are many ways for arranging segments into whole levels and in the context of the framework, the layout algorithm could be viewed as part of the constraints module. To produce the final level, we loop through each location, determine the label based on the sides that are open/closed, sample a latent vector, concatenate the label and forward it through the decoder to generate the segment/room. For the CGMVAE, the directional label is concatenated as in Figure 5. For the CCVAE, both the game blend label and the directional label are concatenated to the sampled latent.

## D. Experiments

For evaluations, we tested that 1) the new directional labels do not prevent the new models from blending in accordance with the blend weights and 2) the generated outputs are open/closed in accordance with the directional labels.

*1) Blend Accuracy:* We performed an identical classifier-based evaluation as for the prior models. We re-used the random forest classifier trained previously for the platformer models and trained a new random forest classifier for the dungeon models. We achieved a 98.8% accuracy on a 80-20 train-test split for the new classifier. Results for platformers and dungeon games are given in Tables VII and VIII respectively. Similar to the prior models, for both platformers and dungeon games, the game classifications using the new models are in accordance with the blend weights in a majority of cases. That is, the game predicted most frequently by the classifier is one whose bit is set to 1 in the binary case. The models do less well for the fractional weights with most predictions being for the game with the highest weight in only 5 out of 8 and 4 out of 6 model-weight pairing for platformers and dungeons respectively with CCVAE doing worse than CGMVAE for dungeons for both types of weights. To compare the two models, we performed the same evaluation as done for the prior models (i.e., the one in Table I) using the same approach described previously. Results are shown in Table IX and confirm that in all but one case, the CGMVAE outperforms the CCVAE, especially when using binary weights. Note that

these results are consistent with the corresponding evaluation for the initial GMVAE and CVAE models, thus suggesting that the additional labels for controlling directionality do not negatively impact the ability of these models to blend games and that like the GMVAE outperforming the CVAE, here the CGMVAE outperforms the CCVAE, with the CCVAE only doing better for fractional weights in the platformer case. This worse performance may stem from the difference in how they enable blending compared to CGMVAE, as previously noted.

| | Platformers | | Dungeons | |
|---|---|---|---|---|
| | Binary | Fractional | Binary | Fractional |
| CGM-32 | **620.27** | 1379.86 | **253.14** | **1820.86** |
| CC-64 | 1282.97 | **1188.01** | 4303.77 | 3014.97 |

TABLE IX: Classification results for blended levels generated by the CGM-32 and CC-64 models. Lower scores are better.

| Genre | Model | Binary | | | Fractional | | |
|---|---|---|---|---|---|---|---|
| | | Exact | Adm | Inadm | Exact | Adm | Inadm |
| Platformers | CGM-32 | 23.03 | 63.4 | 36.6 | 33.34 | 84.41 | 15.59 |
| | CC-64 | 27.05 | 71.01 | 28.99 | 36.24 | 82.28 | 17.72 |
| Dungeons | CGM-32 | 47.33 | 83.77 | 16.23 | 49.67 | 96.29 | 3.71 |
| | CC-64 | 33.43 | 79.31 | 20.69 | 36.77 | 83.54 | 16.45 |

TABLE X: Percentage of exact, admissible (adm) and inadmissible (inadm) matches for models using both weight types.

| | CGM-32 dim | | | | | CCVAE-64 dim | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Blend Weights | SMB | KI | MM | Met | Score | SMB | KI | MM | Met | Score |
| ⟨0001⟩ | 4.7 | 4 | 0.6 | 90.7 | 124.9 | 0.4 | 0 | 0.1 | 99.5 | 0.4 |
| ⟨0010⟩ | 6.5 | 10.6 | 82.8 | 0.1 | 450.5 | 7 | 4 | 89 | 0 | 186 |
| ⟨0011⟩ | 12.5 | 7.1 | 45 | 35.4 | 444.8 | 8.1 | 0.2 | 18.4 | 73.3 | 1607.1 |
| ⟨0100⟩ | 1.9 | 97.3 | 0.6 | 0.2 | 11.3 | 0.1 | 98 | 1.9 | 0 | 7.6 |
| ⟨0101⟩ | 13.9 | 33.6 | 0.9 | 51.6 | 465.5 | 22.5 | 44.2 | 0.1 | 33.2 | 822.1 |
| ⟨0110⟩ | 13 | 40.3 | 46.5 | 0.2 | 275.4 | 25.4 | 32.8 | 41.8 | 0 | 1008.2 |
| ⟨0111⟩ | 15.2 | 26.7 | 28.1 | 30 | 312.5 | 29.8 | 52.3 | 15.6 | 2.3 | 2523.3 |
| ⟨1000⟩ | 53.1 | 46.1 | 0.8 | 1 | 4326.5 | 92.5 | 7.5 | 0.1 | 0 | 112.5 |
| ⟨1001⟩ | 35.9 | 8.6 | 0.6 | 54.9 | 297.1 | 8.6 | 1.1 | 0.7 | 90.3 | 3339.8 |
| ⟨1010⟩ | 31.7 | 13.3 | 54.8 | 0.2 | 534.9 | 62.1 | 22.5 | 15.4 | 0 | 1849.8 |
| ⟨1011⟩ | 33.8 | 6.9 | 35.5 | 23.8 | 143 | 50.2 | 12.3 | 16.7 | 20.8 | 868.7 |
| ⟨1100⟩ | 23.5 | 75.4 | 0.8 | 0.3 | 1348.1 | 58.1 | 41.9 | 0 | 0 | 131.2 |
| ⟨1101⟩ | 36.5 | 34.3 | 0.8 | 28.4 | 35.9 | 19.5 | 75.2 | 0.6 | 4.7 | 2764.4 |
| ⟨1110⟩ | 33.2 | 37.1 | 29 | 0.7 | 33.5 | 22 | 75.6 | 2.4 | 0 | 2871.8 |
| ⟨1111⟩ | 36.1 | 20.6 | 24.4 | 18.9 | 500.1 | 36 | 47 | 7.5 | 9.5 | 1151.5 |
| ⟨0.5, 0.3, 0.2, 0⟩ | 10.7 | 88.7 | 0.5 | 0.1 | 2106.6 | 28.8 | 71.1 | 0.1 | 0.2 | 1221 |
| ⟨0.1, 0.1, 0.1, 0.7⟩ | 15.1 | 14.4 | 0.5 | 70 | 540.7 | 1.6 | 0.2 | 0.1 | 98.2 | 803.8 |
| ⟨0.1, 0.6, 0.2, 0.1⟩ | 2.8 | 95.9 | 1.2 | 0.1 | 1516.3 | 0.6 | 99.4 | 5.3 | 0.2 | 974.5 |
| ⟨0, 0.2, 0.3, 0.5⟩ | 21.6 | 43.9 | 3.5 | 31 | 1355.9 | 11 | 4.2 | 0.4 | 84.4 | 1752.8 |

TABLE VII: Full classification results on platformers. Lower the score, more aligned are predictions with weights.

| | CGM-32 dim | | | | CCVAE-64 dim | | | |
|---|---|---|---|---|---|---|---|---|
| Blend Weights | Zelda | DGG | Met | Score | Zelda | DGG | Met | Score |
| ⟨001⟩ | 0.2 | 0.3 | 99.7 | 0.2 | 0 | 0 | 100 | 0 |
| ⟨010⟩ | 0.1 | 99.8 | 0.1 | 0.1 | 0.1 | 99.8 | 0.1 | 0.1 |
| ⟨011⟩ | 0.3 | 57.8 | 41.9 | 126.5 | 0.7 | 9.9 | 89.4 | 3160.9 |
| ⟨100⟩ | 100 | 0 | 0 | 0 | 1.8 | 70.1 | 28.1 | 15346.9 |
| ⟨101⟩ | 74.3 | 1.8 | 23.9 | 1275 | 0.1 | 0.2 | 99.7 | 4960.1 |
| ⟨110⟩ | 42.1 | 57.6 | 0.3 | 120.3 | 0.2 | 99.4 | 0.4 | 4920.6 |
| ⟨111⟩ | 26 | 46.2 | 27.8 | 250 | 2.4 | 61.1 | 36.5 | 1737.9 |
| ⟨0.5, 0.3, 0.2⟩ | 39.5 | 59.3 | 1.2 | 1322.2 | 0.6 | 68.5 | 30.9 | 4041.4 |
| ⟨0.2, 0.5, 0.3⟩ | 0.4 | 95.3 | 4.3 | 1642.6 | 1 | 78.1 | 20.9 | 1233.4 |
| ⟨0.3, 0.2, 0.5⟩ | 3.7 | 31.4 | 64.9 | 1043.7 | 0.1 | 0.1 | 99.9 | 3770.1 |

TABLE VIII: Full classification results on dungeon games. Lower the score, more aligned are predictions with weights.

*2) Directional Accuracy:* For this, we wanted to test if generated segments had directionality/orientation true to the labels used to generate them. Similar to the blend label evaluation, we trained a random forest classifier on the segments of the three games to predict their directional labels, obtaining 97.13% accuracy using an 80-20 train-test split. For our evaluation, for each generated segment, we compared the label predicted by this classifier with the directional label used to condition its generation. Ideally, we would like the predicted label to exactly match the label used for generation i.e., the generated segment would be open/closed in the precise directions indicated by the label. However, as noted in [8], it is sufficient for the generated segment to match in only the required open directions to ensure playability, irrespective of whether it is open/closed in the desired closed directions. We only really want to avoid the opposite case i.e., the generated segment being closed off in a desired open direction. Borrowing terminology from [8], we use the notion of exact and admissible matches where the former refers to cases where the predicted label exactly

| | CGM-32 dim | | | | CCVAE-64 dim | | | |
|---|---|---|---|---|---|---|---|---|
| Blend Weights | SMB | KI | MM | Met | SMB | KI | MM | Met |
| ⟨0001⟩ | 12.67 | 12.35 | 12.65 | 2.22 | 13.16 | 12.92 | 13.2 | 1.51 |
| ⟨0010⟩ | 10.67 | 10.49 | 2.39 | 11.55 | 11.1 | 10.99 | 1.99 | 11.92 |
| ⟨0011⟩ | 10.39 | 10.23 | 6.61 | 6.94 | 8.52 | 8.61 | 5.55 | 5.69 |
| ⟨0100⟩ | 9.11 | 3.06 | 9.37 | 10.03 | 10.82 | 2 | 10.86 | 11.67 |
| ⟨0101⟩ | 10.58 | 8.95 | 10.67 | 4.28 | 8.63 | 6.95 | 8.74 | 4.07 |
| ⟨0110⟩ | 9.27 | 7.55 | 4.29 | 10.17 | 9.54 | 7.64 | 4.41 | 10.35 |
| ⟨0111⟩ | 9.36 | 8.42 | 6.77 | 6.62 | 5.72 | 5.33 | 3.56 | 5.18 |
| ⟨1000⟩ | 2.88 | 4.1 | 5.06 | 5.68 | 0.94 | 5.26 | 5.12 | 5.7 |
| ⟨1001⟩ | 8 | 8.53 | 8.75 | 2.75 | 6.64 | 7.12 | 7.1 | 1.59 |
| ⟨1010⟩ | 6.12 | 6.68 | 2.29 | 7.42 | 5.17 | 6.06 | 1.97 | 6.53 |
| ⟨1011⟩ | 6.83 | 7.34 | 5.07 | 4.98 | 4.52 | 4.99 | 2.92 | 3.6 |
| ⟨1100⟩ | 5.04 | 3.21 | 6.43 | 7.05 | 3.3 | 2.52 | 4.88 | 5.36 |
| ⟨1101⟩ | 6.94 | 6.52 | 7.82 | 4.14 | 3.45 | 2.97 | 3.88 | 2.62 |
| ⟨1110⟩ | 5.8 | 5.46 | 3.54 | 7.22 | 4.5 | 3.65 | 3.02 | 5.5 |
| ⟨1111⟩ | 6.33 | 6.39 | 5.16 | 5.27 | 3.86 | 3.97 | 2.92 | 3.21 |
| ⟨0.5, 0.3, 0.2, 0⟩ | 4.76 | 3.77 | 5.4 | 6.68 | 3.7 | 3.89 | 4.42 | 5.86 |
| ⟨0.1, 0.1, 0.1, 0.7⟩ | 11.32 | 10.98 | 11.35 | 2.53 | 11.23 | 11.12 | 11.27 | 1.65 |
| ⟨0.1, 0.6, 0.2, 0.1⟩ | 7.46 | 3.74 | 7.42 | 8.38 | 7.92 | 3.08 | 7.05 | 8.75 |
| ⟨0, 0.2, 0.3, 0.5⟩ | 9.42 | 8.82 | 8.65 | 4.29 | 8.88 | 8.81 | 7.77 | 3.62 |

TABLE XI: TPKLDiv values between each original platformer and the different blend configurations. Lower the value, closer the tile-pattern distribution of that blend is to the corresponding game.

matches the conditioning label and the latter to situations where the bits set to 1 in the conditioning label are also 1 in the predicted label, regardless of the value of the bits set to 0 in the conditioning label. Thus, by definition, an exact match is also admissible. For our evaluation, we sampled 1000 latent vectors for each blend weight and conditioned the generation of each using all 15 directional labels. We computed the mean percentage of exact, admissible and inadmissible matches for each weight, averaged across the 15 labels. Results are shown in Table X. While exact matches are not high for any model, the percentage of admissible matches (which includes exact matches) is about 80% in most cases, with the number being lower for the binary platformer models, albeit still much higher than the inadmissible matches. Thus, using the hybrid models, we can both reliably obtain desired blended distributions and also control the direction of segments sampled from them.

*3) Tile Metrics:* We did a TPKLDiv-based evaluation for the new models with results shown in Tables XI and XII. In essentially all cases, games with the highest/joint highest and lowest/joint lowest weight have the smallest/highest TPKLDiv to the corresponding original game respectively. This suggests that, in terms of tile patterns, the blended distributions are closer/farther to the original games in accordance with the blend weights, thus indicating that incorporating directional conditioning to the GMVAE and CVAE does not negatively impact blending, as also suggested by the blending evaluation.

| Blend Weights | CGM-32 dim | | | CCVAE-64 dim | | |
|---|---|---|---|---|---|---|
| | Zelda | DGG | Met | Zelda | DGG | Met |
| ⟨001⟩ | 23.37 | 23.4 | 1.77 | 23.64 | 23.68 | 1.38 |
| ⟨010⟩ | 21.92 | 3.73 | 21.78 | 21.85 | 3.96 | 21.71 |
| ⟨011⟩ | 20.45 | 13.94 | 12.52 | 21.03 | 19.95 | 8.3 |
| ⟨100⟩ | 1.28 | 22.8 | 22.62 | 10.87 | 18.73 | 18.69 |
| ⟨101⟩ | 8.35 | 20.79 | 17.61 | 23.16 | 23.27 | 2.76 |
| ⟨110⟩ | 10.45 | 15.5 | 20.07 | 16.42 | 12.07 | 19.84 |
| ⟨111⟩ | 14.48 | 15.72 | 15.68 | 16.45 | 17.82 | 15.83 |
| ⟨0.5, 0.3, 0.2⟩ | 7.17 | 19.1 | 20.3 | 16.91 | 16.71 | 15.39 |
| ⟨0.2, 0.5, 0.3⟩ | 20.29 | 7.87 | 19.04 | 18.89 | 14.47 | 15.53 |
| ⟨0.3, 0.2, 0.5⟩ | 19.83 | 20.1 | 7.9 | 22.47 | 22.54 | 3.83 |

TABLE XII: TPKLDiv values between each original dungeon game and the different blend configurations. Lower the value, closer the tile-pattern distribution of that blend is to the corresponding game.



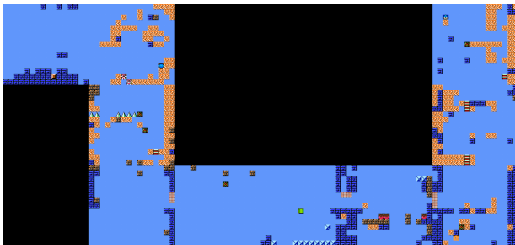Fig. 6: Sample platform level generated using CGMVAE-32



Fig. 7: Sample platform level generated using CCVAE-64

*E. Visual Inspection*

Sample whole levels are shown in Figures 6-9, generated using the layout algorithm described previously with equal weights for each game i.e., 1/4 each for platformers and 1/3 each for dungeons. While these are manually selected, they are representative of the merits of the GMVAE over CVAE in better respecting the blend weights as seen in Figures 6 and 8 which blend content from all platformers and dungeon games while there is little Mario and DGG content (brown tiles) in Figures 7 and 9 respectively. This reflects how the game classifications for the CGMVAE are more spread out in accordance with the weights compared to the CCVAE as shown in Table IX. These visualizations show some amount of noise in generated levels which is expected when blending games. Future work could reduce noise by testing different architectures/training. Our framework is not specific to VAEs so we could try models such as CGANs [24] and GMGANs [34]. We could also conduct user studies to test how preferences and qualitative evaluations match with quantitative evaluations e.g., a model with better metrics might produce levels that a user finds less aesthetically pleasing and vice-versa.

## V. DISCUSSION AND LIMITATIONS

In this section we discuss limitations of this work and some future directions for more thoroughly investigating the nature of blending. We note the classifier-based approach is a proxy for the lack of ground-truth blended levels and recognize its
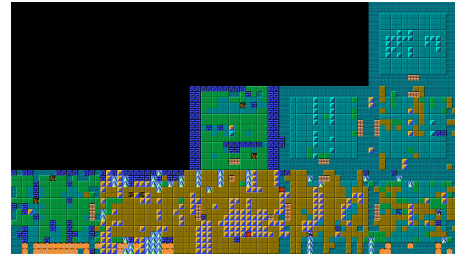


Fig. 8: Sample dungeon level generated using CGMVAE-32



Fig. 9: Sample dungeon level generated using CCVAE-64

issues, e.g., given a blend label ⟨1001⟩, a perfect classification score of 0 can be achieved by a generator outputting a Mario-only level or a Metroid-only level 50% of the time each. While problematic in a vacuum, our models were trained without any signal from a classifier and thus do not learn this adversarial behavior. Combined with the other experiments, the classifier evaluations lend support that the models respect the blend weights. A limitation we do observe is that some games can be poorly represented in the blends, particularly Metroid for platformers and Zelda for dungeons. This may be due to Metroid being the most structurally different from other platformers and thus its latent encodings end up further from those of the other games. When blending the learned game distributions using equal weight, it is possible that the final blend is furthest from the Metroid distribution, leading to fewer Metroid-like segments being sampled. Similarly, for dungeon blends, Zelda with its discrete, self-contained rooms is different than both DGG and Metroid with their more open structures. This could be fixed by scaling the blend weights for these outlier games so that the final blend is closer to the desired combination. Further, we want to more precisely investigate the nature of blending, both in terms of how the levels are blended across games as well as the interaction of level and mechanic blending. Based on evaluations, visual inspection and our intuition, in blending, the models seem to generate game-specific tiles in game-agnostic semantically-equivalent locations in terms of affordances, which is a reasonable analog to how a designer might blend such games. Thus, in addition to our TPKLDiv evaluations, we could evaluate the underlying affordances and compute affordance-based KL-Divergence as in [35]. We could also decouple the level and mechanic blends i.e., test different jump blends on a given level blend and vice-versa, which could enable mixing and matching different level and mechanic blends to produce an even greater variety of games. Relatedly, our approach could

benefit from additional methods of controlling outputs sampled from the blended distribution since currently one can only sample at random. While some control is provided by the CGMVAE and CCVAE in terms of orientation, additional means of control could leverage prior work that combines similar models with evolutionary algorithms to search for specific content [36] or evolve an archive of playable levels [37] based on specified behavioral features. A method for more reliably sampling playable levels is also needed. For this, we could incorporate into the constraints module an A* agent that repairs unplayable levels as in [38].

## VI. CONCLUSION AND FUTURE WORK

We introduced *latent combinational game design*, a framework for controllably blending games, demonstrated it with a supervised GMVAE and a CVAE and proposed a new CGM-VAE architecture which combines the two models to generate whole blended levels. Our future goal is to build a system where the blended games can be played. We also wish to blend mechanics beyond jumps by incorporating affordances and try alternatives besides A* agents for the constraints module, such as an ASP solver. We could also try learning mechanics for a given blended domain by, for example, training an RL agent to learn actions given a set of levels and tile affordances. Future work could also use evolution to search the blended space instead of sampling randomly. Finally, the framework could be used as a combinational creativity approach for artistic domains in general e.g., applied on a dataset of different styles of paintings to produce a custom blend of art styles.

## 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] A. Sarkar, M. Guzdial, S. Snodgrass, A. Summerville, T. Machado, and G. Smith, "Procedural content generation via knowledge transformation (PCG-KT)," *IEEE Transactions on Games*, 2023.

[3] J. Gow and J. Corneli, "Towards generating novel games using conceptual blending," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015.

[4] D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR)*, 2013.

[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 *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020.

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

[8] A. Sarkar and S. Cooper, "Dungeon and platformer level blending and generation using conditional vaes," in *IEEE Conference on Games (CoG)*, 2021, pp. 1–8.

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

[10] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with gaussian mixture variational autoencoders," *arXiv preprint arXiv:1611.02648*, 2016.

[11] R. Shu, "Gaussian mixture VAE: Lessons in variational inference, generative models, and deep nets," 2016. [Online]. Available: http://ruishu.io/2016/12/25/gmvae/

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

[13] M. Guzdial and M. O. Riedl, "Conceptual game expansion," *IEEE Transactions on Games*, vol. 14, no. 1, pp. 93–106, 2021.

[14] S. Snodgrass and S. Ontanon, "An approach to domain transfer in procedural content generation of two-dimensional videogame levels," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016.

[15] Z. Yang, A. Sarkar, and S. Cooper, "Game level clustering and generation using gaussian mixture VAEs," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020.

[16] A. Cao, Y. Luo, and D. Klabjan, "Open-set recognition with gaussian mixture variational autoencoders," in *AAAI Conference on Artificial Intelligence*, 2021.

[17] A. Abdulaziz, J. Zhou, A. Di Fulvio, Y. Altmann, and S. McLaughlin, "Semi-supervised gaussian mixture variational autoencoder for pulse shape discrimination," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

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

[19] G. Fauconnier and M. Turner, "Conceptual integration networks," *Cognitive Science*, vol. 22, no. 2, pp. 133–187, 1998.

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

[21] S. Cooper, "Constraint-based 2D tile game blending in the Sturgeon system," in *EXAG Workshop*, 2022.

[22] A. Summerville, A. Sarkar, S. Snodgrass, and J. Osborn, "Extracting physics from blended platformer game levels," in *EXAG Workshop*, 2020.

[23] A. Summerville, J. Osborn, C. Holmgard, and D. W. Zhang, "Mechanics automatically recognized via interactive observation: Jumping," in *International Conference on Foundations of Digital Games*, 2017.

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

[25] A. Summerville, S. Snodgrass, M. Mateas, and S. Ontañón, "The VGLC: The video game level corpus," in *7th Workshop on PCG at 1st Joint International Conference of DiGRA and FDG*, 2016.

[26] A. Sarkar and S. Cooper, "Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder," in *16th International Conference on the Foundations of Digital Games*, 2021.

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

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[30] A. J. Summerville, S. Philip, and M. Mateas, "MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015.

[31] G. R. Bentley and J. C. Osborn, "The videogame affordances corpus," in *EXAG Workshop*, 2019.

[32] S. M. Lucas and V. Volz, "Tile pattern KL-divergence for analysing and evolving game levels," in *Genetic and Evolutionary Computation Conference*, 2019.

[33] C. Biemer, A. Hervella, and S. Cooper, "Gram-elites: N-gram based quality-diversity search," in *16th International Conference on the Foundations of Digital Games*, 2021.

[34] M. Ben-Yosef and D. Weinshall, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," *arXiv preprint arXiv:1808.10356*, 2018.

[35] A. Sarkar and S. Cooper, "tile2tile: learning game filters for platformer style transfer," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2022.

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

[37] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *IEEE Conference on Games (CoG)*, 2019.

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