Exploring Level Blending across Platformers via Paths and Affordances

Anurag Sarkar[1], Adam Summerville[2], Sam Snodgrass[3], Gerard Bentley[4] and Joseph Osborn[4]
[1]Northeastern University
[2]California State Polytechnic University
[3]modl.ai
[4]Pomona College
sarkar.an@northeastern.edu, asummerville@cpp.edu, sam@modl.ai, gbkh2015@mymail.pomona.edu, joseph.osborn@pomona.edu

Abstract

Techniques for procedural content generation via machine learning (PCGML) have been shown to be useful for generating novel game content. While used primarily for producing new content in the style of the game domain used for training, recent works have increasingly started to explore methods for discovering and generating content in novel domains via techniques such as level blending and domain transfer. In this paper, we build on these works and introduce a new PCGML approach for producing novel game content spanning multiple domains. We use a new affordance and path vocabulary to encode data from six different platformer games and train variational autoencoders on this data, enabling us to capture the latent level space spanning all the domains and generate new content with varying proportions of the different domains.

Introduction

Procedural content generation via machine learning (PCGML) (Summerville et al. 2018) refers to techniques for PCG using models trained on existing game data. This enables the generation of game content that is novel but still captures the characteristics and patterns of the data used for training. While a lot of prior PCGML work has looked at generating content for a single domain such as *Super Mario Bros.* (Summerville and Mateas 2016), *The Legend of Zelda* (Summerville and Mateas 2015) and *Doom* (Giacomello, Lanzi, and Loiacono 2018), recent works have started to focus on more creative PCGML techniques (Guzdial and Riedl 2018b) that attempt to learn models capable of producing content outside of the domain used for training. That is, rather than attempting to produce novel variations of content within an existing domain, these techniques blend existing domains into entirely new ones and generate content for these new domains. This yields models that can generalize better across different domains—a key challenge of PCGML. Moreover, such cross-domain models could prompt the discovery of novel game designs previously hidden in the combined design space of the input domains. Techniques for such PCGML approaches have involved domain transfer (Snodgrass and Ontañón 2016), blending separate models to produce generators (Sarkar and Cooper 2018), and constructing game graphs using learned models of levels and game rules (Guzdial and Riedl 2018a). Our paper most directly builds on the approach of training models on multiple domains to learn a new blended domain space (Sarkar, Yang, and Cooper 2019).

We extend the prior method in a number of ways. We increase the input domain from two to six games thus greatly expanding the possibility space of the learned, blended domain in terms of new content. We also introduce a new, unified affordance vocabulary consistent across all domains which expands significantly on such vocabularies used in prior PCGML work. Rather than ignoring game mechanics, we annotate the input domains with paths generated by an A* agent informed by the jump arcs of each domain. Finally, like prior blending work, we use variational autoencoders (VAEs) to learn our models but train two varieties—one composed of linear layers and another using GRU layers—and compare their performance. To the best of our knowledge, this is the first application of a GRU-VAE for PCGML.

Related Work

Procedural content generation via machine learning (PCGML) (Summerville et al. 2018) describes a collection of PCG techniques that leverage existing data to build a model from which new content can be sampled. There has been extensive work on developing and adapting various models (e.g., LSTMs (Summerville and Mateas 2016), GANs (Volz et al. 2018), Bayesian Networks (Guzdial and Riedl 2016), etc.) to be used in specific domains (e.g., *Super Mario Bros.* (Snodgrass and Ontañón 2017a), *The Legend of Zelda* (Summerville and Mateas 2015), *Kid Icarus* (Snodgrass and Ontañón 2017a), etc.). The drawback of these PCGML approaches is that they require training data from the domain they are meant to create content for, limiting their use in new domains.

Several researchers have considered the limitation of requiring training data in the target domain. Snodgrass and Ontañón (2016) presented a domain transfer approach that tried to find mappings between domains to enable data in one domain to supplement the training data in another. This still requires some data from a target domain and cannot

generate content for new domains. We instead extend prior work that blends domains together to produce new domains. Sarkar and Cooper (2018) used LSTMs trained on a combined dataset as well as weighting separately trained LSTMs to generate blended levels with segments from both *Super Mario Bros.* and *Kid Icarus*. Sarkar, Yang, and Cooper (2019) further applied VAEs for level blending, training models on a combined dataset of segments from *Mario* and *Kid Icarus* to admit interpolation between these domains. We extend the above blending and transfer approaches by: incorporating several additional domains to blend; defining a unified level representation for the domains; annotating level data with path information (so models can train on gameplay as well as structural information); and evaluating a GRU-VAE approach to level blending.

Other approaches have also been considered for blending games. Gow and Corneli (2015) presented a framework for blending VGDL (Schaul 2014) descriptions of games, demonstrating their approach by creating *Frolda*—a game created by blending *Zelda* and *Frogger*. However, their approach was manual and limited to the VGDL framework. Guzdial and Riedl (2018a) presented *conceptual expansion*—an ML-based approach which creates game concept graphs that can be blended and recombined to produce new games. Recently, Snodgrass and Sarkar (2020) combined binary space partitioning and VAEs to generate blended platformer levels using a sketch-based level representation.

Like some past blending works, we use variational autoencoders (VAEs). VAEs (Kingma and Welling 2013), as well as vanilla autoencoders (Hinton and Salakhutdinov 2006), have been used in several prior PCGML works. Jain et al. (2016) used autoencoders for generating and repairing Mario levels while Guzdial et al. (2018) used autoencoders to generate Mario level structures conforming to specific design patterns. Thakkar et al. (2019) applied both vanilla and variational autoencoders for generating *Lode Runner* levels. Aside from level generation, Alvernaz and Togelius (2017) used an autoencoder to learn a low-dimensional representation of the *VizDoom* environment and used it to evolve controllers while Soares and Bulitko (2019) used VAEs for classifying procedurally generated NPC behavior.

Level Representation

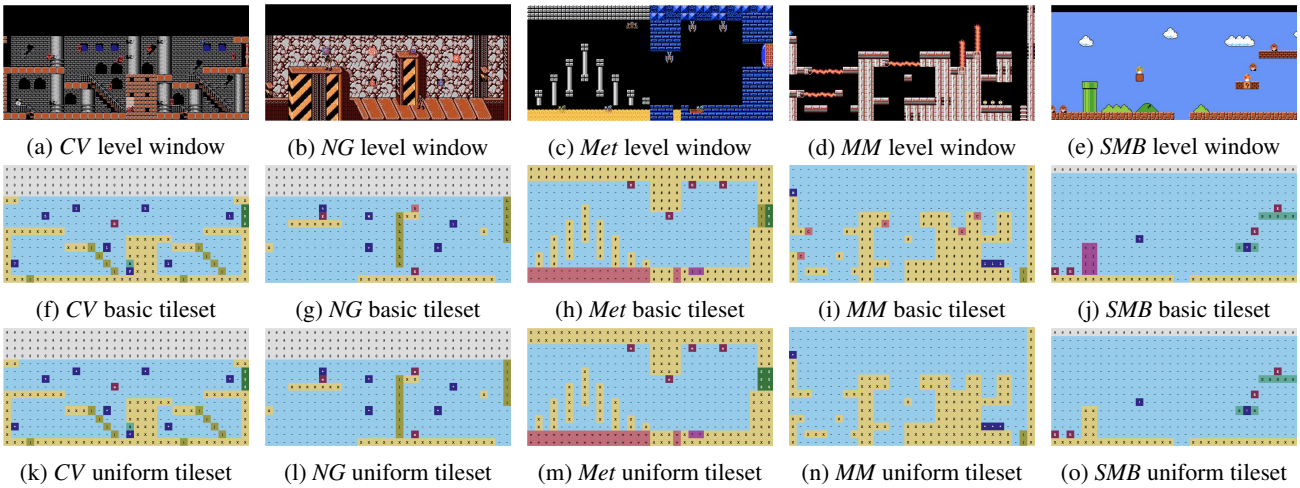
We evaluate our approaches using six classic NES platforming games, namely, *Super Mario Bros.*, *Super Mario Bros. II: The Lost Levels*, *Ninja Gaiden*, *Metroid*, *Mega Man*, and *Castlevania*. Levels in these games vary greatly in their specific tile-based representations commonly used by PCGML techniques (Summerville et al. 2018) and the VGLC (Summerville et al. 2016) (e.g., different tile types defined for enemy types, power-ups, obstacles, etc.). For our models to reason across domains, we need to unify the representations. We accomplish this by leveraging the tile affordance work of Bentley and Osborn (2019) to derive a common language.

We describe tiles in terms of the following affordances: *solid* (the player can stand on this tile); *climbable* (the player can use this tile to climb); *passable* (the player can pass through this tile); *powerup* (this tile strengthens the player in some way); *hazard* (this tile harms the player); *moving* (this tile changes location); *portal* (this tile transports the player somewhere); *collectable* (the player can pick up this tile); *breakable* (the player can destroy this); and *null* (this tile indicates a position outside of the actual level geometry). This differs from the Bentley and Osborn affordances and is specialized for side-scrolling games. Using the above affordances, we defined a uniform set of tile types to represent all of the domains in our experiments:

- `X`: *solid*, (e.g., ground or platforms)
- `S`: *solid, breakable*, (e.g., breakable bricks in *SMB*)
- `#`: *solid, moving*, (e.g., moving platforms)
- `|`: *solid, passable, climbable*, (e.g., ladders)
- `v`: *hazard*, (e.g., spikes)
- `^`: *solid, hazard*, (e.g., lava or solid spikes)
- `e`: *moving, hazard*, (e.g., enemies)
- `E`: *solid, moving, passable, hazard*, (e.g., enemies the player could pass through or jump on)
- `o`: *collectable*, (e.g., coins)
- `*`: *collectable, powerup*, (e.g., weapon refills in *MM*)
- `Q`: *solid, collectable*, (e.g., coin blocks in *SMB*)
- `!`: *solid, powerup*, (e.g., mushroom blocks in *SMB*)
- `$`: *portal*, (e.g., doors in *Metroid*)
- `@`: *solid, null, hazard*

We further annotate our levels with A^* agent-generated paths, where the agent is tuned to each domain prior to annotation according to the possible jump arcs in that domain. We annotate our levels with paths because 1) incorporating pathing information has been shown to be beneficial to PCGML techniques when learning the level geometry (Summerville and Mateas 2016; Snodgrass and Ontañón 2017b), and 2) it is our longer term goal to learn and blend not only the level geometry, but the jump physics as well.

Lastly, the levels in each domain have vastly different sizes and dimensions. We address this by breaking each level into 15×32 sized segments, padding them vertically when needed. For this work, we used the horizontal portions of these levels to enable more holistic blending across domains and leave considerations of vertical level segments in *Ninja Gaiden*, *Metroid* and *Mega Man* for future work. Thus, to generate our training data, we slid a 15×32 window horizontally across all levels. Since some of the games have discrete rooms (*Castlevania*, *Mega Man*, *Ninja Gaiden*, and *Metroid*), if a door is found in the segment and it does not lie on the edge of the segment, the segment is discarded. Duplicate segments are also discarded (these often appear in *Metroid*). We produced 775 segments for *Castlevania (CV)*, 1907 segments for *Mario (SMB)* (henceforth, we use *Mario* and *SMB* to refer to the combined domain of both *Super Mario Bros.* and *Super Mario Bros II: The Lost Levels*), 924 segments for *Mega Man (MM)*, 1833 segments for *Metroid (Met)* and 504 segments for *Ninja Gaiden (NG)*. We oversampled all domains other than *SMB* to obtain approximately equal number of segments from all games in order to prevent the learned blended models from skewing towards any specific game(s).



(a) *CV* level window (b) *NG* level window (c) *Met* level window (d) *MM* level window (e) *SMB* level window

(f) *CV* basic tileset (g) *NG* basic tileset (h) *Met* basic tileset (i) *MM* basic tileset (j) *SMB* basic tileset

(k) *CV* uniform tileset (l) *NG* uniform tileset (m) *Met* uniform tileset (n) *MM* uniform tileset (o) *SMB* uniform tileset

Figure 1: This figure shows a window from a level from each of our domains (top row), that window represented using that domain's defined tileset (center row), and that window represented using our uniform tileset (bottom row).

Modeling and Generation

We used two different variants of VAEs for building our models. Both were implemented using PyTorch (Paszke et al. 2017) and are described in the following sections.

Linear VAE

For the linear model, the encoder and decoder each consisted of 4 fully-connected linear layers using ReLU activation. Encoder layers had dimensionalities (11520x1024), (1024x512), (512x256) and (256xlatent size) with decoder layers having these in reverse order. We trained four versions of this model using latent dimensions of 32, 64, 128 and 256. All models were trained for 5000 epochs using the Adam optimizer and a learning rate of 0.001.

GRU-VAE

The GRU-VAE is a sequence-to-sequence recurrent architecture that has encoder and decoder Gated Recurrent Units (GRU) (Chung et al. 2014) with a variational latent sampling. All GRU-VAEs were trained with the same numbers and sizes of recurrent layers, with the only differences being the latent dimensions of 32, 64, 128 and 256—similar to the Linear VAE. As in the earlier work of Summerville and Mateas (2016), a top-to-bottom approach is used to turn the 2D levels into a 1D sequence. The encoder had 3 hidden layers of size 1024, and the decoder had 2 hidden layers of size 256—both had a dropout rate of 50%. To aid in the convergence of the model, the variational loss was annealed with a linear rate from 0 to 0.05 times the variational loss over 5 epochs before the rest of the training continued at that rate—for a total of 50 epochs using the Adam optimizer and a learning rate of $1e-5$. Note that the GRU-VAE is non-deterministic in its decoding. At decoding time, the decoder is initialized with the latent embedding and then proceeds to decode in an auto-regressive manner with sampling. For each generation, 10 segments are sampled and the one with the lowest perplexity (highest likelihood) is kept.

Evaluation

Tile-based Metrics

For each model, we generated 1000 segments and evaluated them against the training data with the following metrics:

- *Density*: proportion of a segment not occupied by background or path tiles
- *Non-Linearity*: how well the topology of a segment fits to a line; measured by fitting a line to the topmost point of columns in a segment using linear regression
- *Leniency*: the proportion of a segment not occupied by hazard tiles; this acts as a very simple proxy for difficulty
- *Interestingness*: the proportion of a segment occupied by powerups, portals and collectables
- *Path-Proportion*: the proportion of a segment occupied by the optimal path through it

For our evaluations, we computed the *E-distance* (Székely and Rizzo 2013) which measures the distance between two distributions. Summerville (2018) suggests E-distance as a metric for evaluating generative models. Lower values of E-distance imply higher similarity between distributions. We calculated E-distance with the above metrics for the generated segments against the training segments, per domain. The 5 metrics were combined into a 5-dimensional feature vector for each distribution and the E-distance was computed using these combined features.

Agent-based Evaluation

We tested our models' ability to blend domains by interpolating between level segments. For this, we randomly selected 10 level segments from each domain, passed each segment through the encoder of the VAE to get the latent vector representation of that segment, and then interpolated between such latent vectors in increments of 25% (e.g., 25% domain A, level n and 75% domain B, level m). We

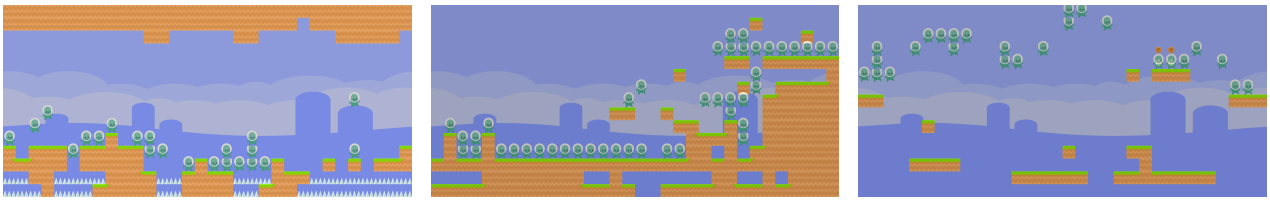


Figure 2: Linear Samples

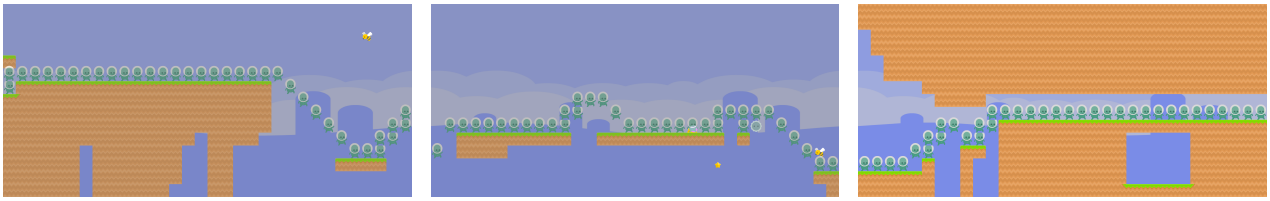


Figure 3: GRU Samples

Model	ALL	CV	MM	Met	SMB	NG
LIN32	*0.58*	1.66	*3.53*	12	4.76	3.88
-64	0.6	0.9	5.52	*10.6*	3.1	2.54
-128	0.99	*0.86*	6.58	13.72	3.25	*2.29*
-256	0.88	0.97	5.92	14.14	*3.00*	2.82
GRU32	2.28	1.22	4.16	**0.4**	0.38	**1.22**
-64	1.25	1.15	**2.4**	0.89	**0.34**	1.9
-128	0.46	0.76	5.08	2.45	2.07	1.36
-256	**0.32**	**0.66**	4.81	0.52	3.13	1.78

Table 1: E-distances between each model and all games together as well as each game taken separately. For both cases, $p < .01$ for all models, using 100 resamples. For separate games, *Italics* denote the best for the specific NN architecture, and **Bolds** denote the best across all architectures.

then generated level segments by forwarding interpolated latent vectors through the VAE decoder. Recall that the models generate sections annotated with beliefs about the path through the section. To evaluate an interpolated section, we compared the generated path to the paths found by A* agents for each of the blended domains by computing the discrete Fréchet distance (Eiter and Mannila 1994) between the generated paths and the agents' paths. The Fréchet distance has been used previously to measure the similarity of agent paths (Snodgrass and Ontañón 2017b); it can be thought of as the length of rope needed to connect two people walking on separate paths over the entirety of the paths. Agents that cannot traverse the entire level segment are said to have failed and their attempts are not included in this metric.

The Fréchet distance between the generated and agent paths in interpolated segments gives insight into how well such segments capture both pathing and structural information in the component domains. Intuitively, if a segment is meant to be 75% *MM* and 25% *NG*, we expect the *MM* A* agent to find a more similar path to the generated path. Additionally, we can examine which domains our model struggles with by comparing the distances across domains e.g., determining if *SMB* A* paths are typically more similar to the generated paths than *Met* A* paths.

Results

The E-distances between the samples from each of the generators and the original games taken together and separately are shown in Table 1. The distributions for the generated segments were all statistically different from the original games ($p < 0.01$) meaning that no generator successfully replicated the expressive range of the original levels; however, we see that when compared across all games, the linear models generally have lower distances, although the two largest GRU models surpass all linear models. We also see how each generator does when primed to generate segments that are targeted to be like a specific game, done by sampling the latent space using the means and variances of the latent encodings of the segments for that game. In this case, the GRU models are generally closer to the original games than the linear models, although no one model dominates. For e.g., GRU-32 and GRU-64 both outperform all linear models for *Met*, *SMB* and *NG* with GRU-64 also doing better than all linear models for *MM* but both these GRU models do worse than 3 out of the 4 linear models for *CV*. Similarly, we do not see any particular advantage for latent size when compared using games separately, unlike when compared against all taken together. The GRU-32 and GRU-64 models are tied for the most games that they are closest to, but GRU-256 still beats both for *CV*.

A key contribution of this work is the inclusion of agent paths across multiple games, with the goal being the development of novel physics models for blended content. Toward that end, we assess how well agents designed for the input games perform on levels created by these models. Table 2 shows the Fréchet distance between the generated path from game-specific samples and agents capable of playing that game. The agent failure rate is the percentage of generated levels that the A* agent failed to complete. The GRU models do better than the linear models in generating paths that are more similar to those in the target domains—with a commensurate decrease in agent failure rate. This is ex-

Model	Domain CV	MM	Met	SMB	NG	Agent Failure Rate
LIN-32	$4.12 \pm (2.66)$	$4.83 \pm (2.38)$	$\mathit{4.59} \pm (\mathit{2.15})$	$5.24 \pm (2.75)$	$4.56 \pm (2.67)$	11.46%
LIN-64	$4.12 \pm (2.49)$	$4.85 \pm (2.49)$	$4.69 \pm (2.16)$	$5.23 \pm (2.75)$	$\mathit{4.41} \pm (\mathit{2.52})$	11.72%
LIN-128	$\mathit{3.87} \pm (\mathit{2.59})$	$\mathit{4.70} \pm (\mathit{2.49})$	$4.60 \pm (2.38)$	$5.12 \pm (2.91)$	$4.44 \pm (2.61)$	*11.25%*
LIN-256	$4.06 \pm (2.69)$	$4.89 \pm (2.54)$	$4.74 \pm (2.38)$	$\mathit{5.01} \pm (\mathit{2.87})$	$4.54 \pm (2.65)$	11.14%
GRU-32	$\mathbf{1.81} \pm (\mathbf{1.99})$	$2.18 \pm (2.00)$	$\mathbf{1.98} \pm (\mathbf{1.74})$	$2.43 \pm (2.14)$	$\mathbf{1.81} \pm (\mathbf{1.85})$	**4.52%**
GRU-64	$2.30 \pm (2.34)$	$2.65 \pm (2.53)$	$2.27 \pm (2.17)$	$3.25 \pm (2.76)$	$2.19 \pm (2.28)$	5.12%
GRU-128	$1.94 \pm (2.15)$	$\mathbf{1.97} \pm (\mathbf{2.25})$	$2.13 \pm (2.13)$	$\mathbf{2.02} \pm (\mathbf{2.16})$	$1.98 \pm (2.14)$	4.81%
GRU-256	$2.10 \pm (2.22)$	$2.24 \pm (2.48)$	$2.38 \pm (2.25)$	$2.26 \pm (2.30)$	$2.18 \pm (2.27)$	4.55%

Table 2: Fréchet distances for each model on each domain. Columns include blended levels using that domain. Values in cells are the average Fréchet distance between the generated paths in the blended levels and the agent paths for the blended domains.

GRU-32	CV	MM	Met	SMB	NG
CV	$1.76 \pm (2.38)$	$1.95 \pm (2.12)$	$1.63 \pm (1.70)$	$2.18 \pm (2.19)$	$1.51 \pm (1.73)$
MM	$1.95 \pm (2.12)$	$2.45 \pm (1.83)$	$2.02 \pm (1.67)$	$2.92 \pm (2.36)$	$1.81 \pm (1.63)$
Met	$1.63 \pm (1.70)$	$2.02 \pm (1.67)$	$2.21 \pm (1.30)$	$2.41 \pm (1.70)$	$1.74 \pm (1.88)$
SMB	$2.18 \pm (2.19)$	$2.92 \pm (2.36)$	$2.41 \pm (1.70)$	$2.69 \pm (2.24)$	$2.10 \pm (20.12)$
NG	$1.51 \pm (1.73)$	$1.81 \pm (1.63)$	$1.74 \pm (1.88)$	$2.10 \pm (20.12)$	$1.85 \pm (1.51)$
LIN-128	**CV**	**MM**	**Met**	**SMB**	**NG**
CV	$2.79 \pm (2.22)$	$4.10 \pm (2.53)$	$3.92 \pm (2.43)$	$4.14 \pm (2.84)$	$3.57 \pm (2.53)$
MM	$4.10 \pm (2.53)$	$3.64 \pm (1.98)$	$4.56 \pm (2.07)$	$5.81 \pm (2.80)$	$4.57 \pm (2.25)$
Met	$3.92 \pm (2.43)$	$4.56 \pm (2.07)$	$3.70 \pm (1.55)$	$5.30 \pm (2.61)$	$4.71 \pm (2.35)$
SMB	$4.14 \pm (2.84)$	$5.81 \pm (2.80)$	$5.30 \pm (2.61)$	$5.62 \pm (3.08)$	$5.10 \pm (3.01)$
NG	$3.57 \pm (2.53)$	$4.57 \pm (2.25)$	$4.71 \pm (2.35)$	$5.10 \pm (3.01)$	$3.31 \pm (2.34)$

Table 3: Fréchet distances for pairs of blended domains for the best performing Linear (LIN-128) and GRU model (GRU-32).

pected as a key advantage of recurrent models is the memory during decoding which allows them to remember where they have generated paths so far, thus better generating connected, sensible paths. The best performing model is GRU-32 with the lowest failure rate and the lowest Fréchet distances for three of the five games. The best performing linear model is LIN-128 with the lowest failure rate amongst the linear models and the lowest scores for two of five games.

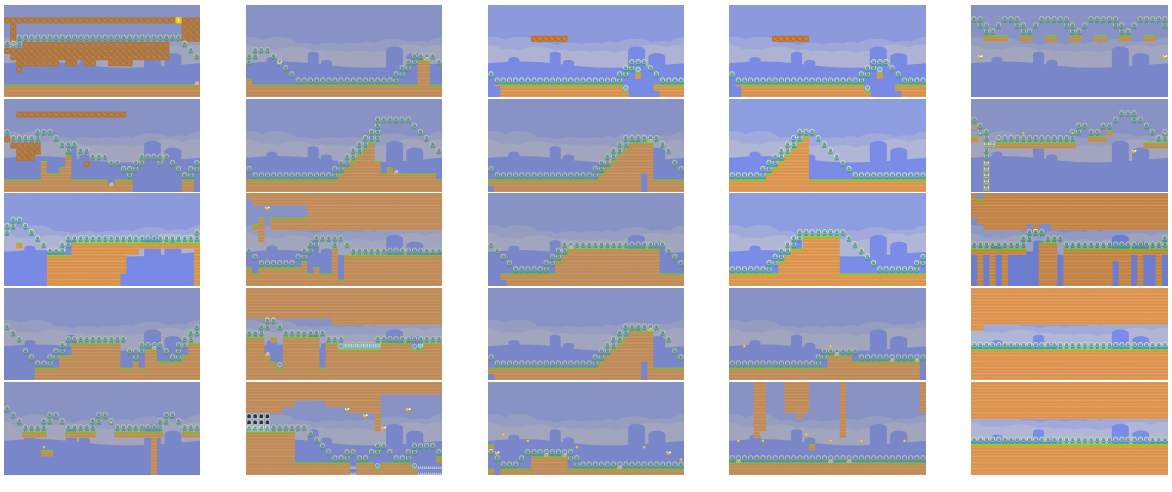
Fréchet distances for blended levels generated by GRU-32 and LIN-128 are shown in Table 3. Distances for the GRU model are lower than the linear model, across the board; however, we see some commonalities. The distances for games blended with *SMB* are higher than other games across both models. Additionally, the blends for both *CV* and *NG* produce lower distances across models—both have more predictable jump physics (players do not control jump height) so their physics models may be simpler to learn.

Examples of randomly sampled levels generated using both models are shown in Figures 2 and 3 using a neutral sprite representation for all games (taken from Kenney[1]). Both the linear and the GRU models are able to learn different "styles" of levels—e.g., some are cave-like while some are more open. The GRU model produces contiguous paths while the linear model's paths are noisier and less coherent. Example interpolations between all pairs of games using the GRU model are shown in Figure 4. Figure 5 shows a single such interpolation using the linear model. The GRU model has a number of segments that it finds highly likely—e.g., a flat section with a hill (SMB ↓ Met, NG, CV) or a long cavelike corridor (CV ↓ MM, Met). The autoregressive generation of the GRU is possibly a reason such segments show up—it tries to produce sequences that are likely based on what has been generated so far. The linear model's output looks much closer to a straight interpolation of the two segments (tiles that appear in one and are empty in the other disappear in the interpolation) with the patterns that emerge looking noisier than the endpoints of the interpolation. The GRU model's interpolations are more coherent with none of the noise of the linear model; however, the interpolations are also less obviously interpolations. Many interpolations make sense: SMB ↓ MM goes from a lot of destructible blocks and little empty space to the opposite; CV ↓ MM goes from a wide-open space to a cave-like hallway with the interpolations getting more and more filled in; and MM ↓ Met goes from wide open to more constrained with doors, and the 50% mark looks like a compromise of the two. On the other hand, the interpolations between SMB ↓ NG, SMB ↓ CV, and Met ↓ NG seem to be determined by their respective endpoints. MM ↓ NG has very little variation in the interpolations, perhaps due to the end point similarity.

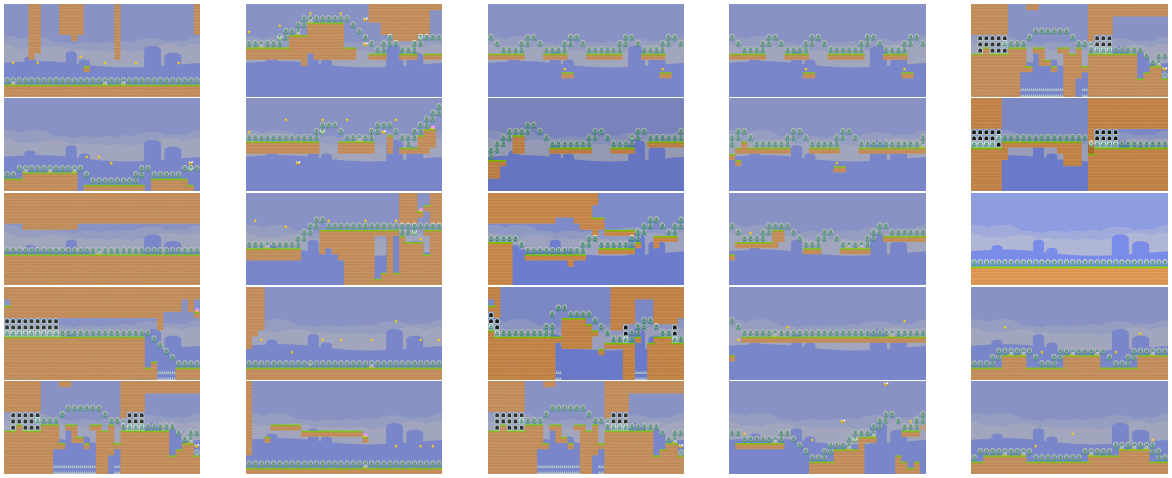
Conclusion and Future Work

We presented a new PCGML approach that leverages path information and a new affordance vocabulary to extend existing VAE-based level generation and blending techniques to produce traversable blended levels across a greater number of game domains. In the future, our approach to generate traversable level blends by incorporating paths from multi-

[1]https://www.kenney.nl/assets/platformer-art-deluxe



(a) GRU SMB ↓ MM (b) GRU SMB ↓ Met (c) GRU SMB ↓ NG (d) GRU SMB ↓ CV (e) GRU CV ↓ MM



(f) GRU CV ↓ Met (g) GRU CV ↓ NG (h) GRU MM ↓ Met (i) GRU MM ↓ NG (j) GRU Met ↓ NG

Figure 4: Example interpolations for all pairs of games.

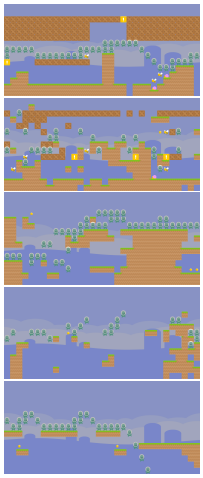


Figure 5: Linear SMB ↓ MM

ple domains could enable learning blended physics models spanning multiple games. Physics models are useful for informing game-playing agents such as the A* agents used to encode paths in this work. Thus, learning and extracting such models across multiple game domains would complement the multi-domain blended levels presented above.

We limited our approach to horizontal level sections in this work. Future work can explore blending vertical sections from *Metroid*, *Mega Man* and *Ninja Gaiden* along with games like *Kid Icarus*. Blending levels and physics for horizontal and vertical domains will support future work in blending games that simultaneously scroll in two directions. Finally, while prior PCGML works have looked at other game genres, techniques like blending and domain transfer have not been applied outside of platformers. Future work should test such approaches in other genres such as action-adventure games and dungeon crawlers.

References

Alvernaz, S., and Togelius, J. 2017. Autoencoder-augmented neuroevolution for visual doom playing. In *IEEE Conference on Computational Intelligence in Games*.

Bentley, G. R., and Osborn, J. C. 2019. The videogame affordances corpus. *2019 Experimental AI in Games Workshop*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Eiter, T., and Mannila, H. 1994. Computing discrete Fréchet distance. Technical Report 94/64, Technische Universität Wien.

Giacomello, E.; Lanzi, P. L.; and Loiacono, D. 2018. Doom level generation using generative adversarial networks. In *IEEE Games, Entertainment, Media Conference (GEM)*.

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

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

Guzdial, M., and Riedl, M. 2018a. Automated game design via conceptual expansion. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Guzdial, M., and Riedl, M. 2018b. Combinatorial creativity for procedural content generation via machine learning. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

Guzdial, M.; Reno, J.; Chen, J.; Smith, G.; and Riedl, M. 2018. Explainable PCGML via game design patterns. In *Proceedings of the AIIDE Workshop on Experimental AI in Games*.

Hinton, G., and Salakhutdinov, R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

Jain, R.; Isaksen, A.; Holmgard, C.; and Togelius, J. 2016. Autoencoders for level generation, repair and recognition. In *ICCC Workshop on Computational Creativity and Games*.

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

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

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

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

Schaul, T. 2014. An extensible description language for video games. *IEEE Transactions on Computational Intelligence and AI in Games* 6(4):325–331.

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

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

Snodgrass, S., and Ontañón, S. 2017b. Procedural level generation using multi-layer level representations with mdmcs. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 280–287. IEEE.

Snodgrass, S., and Sarkar, A. 2020. Multi-domain level generation and blending with sketches via example-driven bsp and variational autoencoders. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*.

Soares, E. S., and Bulitko, V. 2019. Deep variational autoencoders for npc behaviour classification. In *IEEE Conference on Games*.

Summerville, A., and Mateas, M. 2015. Sampling hyrule: Sampling probabilistic machine learning for level generation. *Proceedings of the Foundations of Digital Games*.

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

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

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

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

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

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

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 221–228. ACM.