Blending Levels from Different Games using LSTMs

Anurag Sarkar and **Seth Cooper**
Northeastern University, Boston, Massachusetts, USA
sarkar.an@husky.neu.edu, scooper@ccs.neu.edu

Abstract

Recent work has shown machine learning approaches to be effective in training models on existing game data for informing procedural generation of novel content. Specifically, ML techniques have successfully used existing game levels to train models for generating new levels and blending existing ones. Such blending of not just levels, but entire games, has also been proposed as a possible means of generating new games. In this paper, we build on these works by training Long Short Term Memory recurrent neural networks on level data for two separate platformer games—*Super Mario Bros.* and *Kid Icarus*—and use these models to create new levels that are blends of levels from both games, thus creating a level generator for a novel, third game.

Introduction

Procedural content generation (PCG) is the automatic creation of game content via procedural or algorithmic methods (Shaker, Togelius, and Nelson 2016). Since its use in games like *Elite* (Braben and Bell 1984) and *Rogue* (A.I. Design 1980), PCG has been widely adopted for generating levels, maps, weapons, rules, terrain, etc., and has been an active field of games research. Traditional PCG methods include search-based optimization (Togelius et al. 2011), constraint satisfaction (Smith and Mateas 2011) and grammar-based methods (Smith, Whitehead, and Mateas 2011), to name a few. However, these often rely on designer-authored rules, parameters and constraints to guide the generation process and ensure that the generated content has desired properties and characteristics. Aside from being time consuming, such methods may unintentionally capture designer biases. PCG via Machine Learning (PCGML) has thus emerged as an alternative to the traditional methods to help overcome their limitations. Summerville et al. (2017) define PCGML as "the generation of game content using models that have been trained on existing game content." By training on game data that one wishes to emulate or create novel variations of, one can capture the desired properties within the trained model and sample from it to generate new content.

Recent work has shown that models trained on existing *Super Mario Bros.* levels can generate new levels via both sequence prediction (Summerville and Mateas 2016) as well as blending existing levels (Guzdial and Riedl 2016b). Moreover, Gow and Corneli (2015) proposed a theoretical framework for generating new games by blending not just levels but entire games, showing that it is possible to blend two games to create a third whose aesthetics and mechanics are combinations of those of the original two games.

In this work, we take a step towards implementing this framework by leveraging PCGML and concept blending. Specifically, we train Long Short Term Memory recurrent neural networks (LSTMs) on existing levels of the platformers *Super Mario Bros.* and *Kid Icarus*. We then sample from the trained models to generate new levels that encode structural characteristics and properties of levels from both games. We thus create a level generator that can produce levels for a third game whose levels contain properties of levels from the two games used for training. We also implement a parametrized version of the generator that allows a designer to control the approximate amount of each original game desired in the final blended levels.

Related Work

PCG via Machine Learning (PCGML). PCGML has emerged as a promising research area as evidenced by many successful applications of ML for content generation. Neural networks in particular have seen wide use for level and map generation. Hoover, Togelius, and Yannakakis (2015) generated *Super Mario Bros.* levels by combining a music composition technique (*functional scaffolding*) with a method for evolving ANNs (*NeuroEvolution of Augmenting Topologies* (Stanley and Miikkulainen 2002)). Autoencoders have also found related applications, with Jain et al. (2016) training one on existing levels to generate new levels and repair generated levels that were unplayable. N-gram models have also been used by Dahlskog, Togelius, and Nelson (2014) for generating *Super Mario Bros.* levels. Guzdial and Riedl (2016a) used probabilistic graphical models and clustering to create levels for *Super Mario Bros.* by training on gameplay videos. Besides platformers, Summerville et al. (2015) used Bayes Nets for creating *The Legend of Zelda* levels. They also used Principal Component Analysis to interpolate between existing Zelda dungeons to create new ones (Summerville and Mateas 2015).

Markov models have also found use in generating content. Snodgrass and Ontañón have done extensive work in

generating levels for *Super Mario Bros.*, *Kid Icarus* and *Lode Runner* (Broderbund 1983) using multi-dimensional Markov chains (Snodgrass and Ontañón 2014), with hierarchical (Snodgrass and Ontañón 2015) and constrained (Snodgrass and Ontañón 2016) extensions as well as using Markov random fields. In particular, our work in blending levels is most similar to their work on domain transfer using Markov chains for mapping levels from one game to another (Snodgrass and Ontañón 2017).

Finally, Long Short Term Memory recurrent neural networks (LSTMs) have been particularly effective in generating game levels. Summerville and Mateas (2016) used LSTMs to generate Mario levels by treating each level as a character string and using these strings to train the network. The trained network could then use an initial string as a starting seed to generate new characters, thereby producing new levels. Though most level generation using LSTMs focuses primarily on *Super Mario Bros.*, the method can be used to create generators for any game whose levels are represented as sequences of text. The success of such techniques for level generation informed our use of LSTMs in this work.

Mixed-Initiative PCG. This refers to content generation by harnessing the capabilities of a procedural generator and a human designer working in concert. Such generators, like *Tanagra* (Smith, Whitehead, and Mateas 2011), combine the generator's ability to rapidly produce multiple levels with the human ability to evaluate them using superior creativity and judgment. Authorial control in procedural generators allows designers to guide generation towards desired content. Thus, we implemented a parametrized variant of the blended level generator which lets the designer control the percentage of either game in the final blend. Yannakakis, Liapis, and Alexopoulos (2014) offer an extensive analysis of mixed-initiative design tools and their effects on creativity.

Concept Blending. Concept blending states that novel concepts can be produced by combining elements of existing concepts. The "four space" concept blending theory was proposed by Fauconnier and Turner (1998; 2008) and describes a conceptual blend as consisting of four spaces:

- Two *input spaces* constituting the concepts prior to being combined
- A *generic space* into which the input concepts are projected and equivalence points are identified
- A *blend space* into which the equivalent points from the generic space are projected and new concepts are evolved

Goguen (1999) offers a related formalization of conceptual blending which forms the basis of the COINVENT computational model (Schorlemmer et al. 2014). This aims to develop a "computationally feasible, formal model of conceptual blending" and has applied conceptual blending in music (Cambouropoulos, Kaliakatsos-Papakostas, and Tsougras 2014) and mathematics (Bou et al. 2015).

Blending Game Levels and Game Generation. In games, Guzdial and Riedl (2016b) used conceptual blending to blend level generation models of *Super Mario Bros.* and also looked at different blending approaches to generate levels (Guzdial and Riedl 2017). Additionally, Gow and Corneli (2015) proposed applying conceptual blending not just to

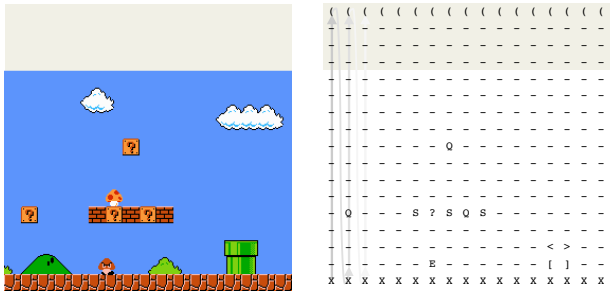

Figure 1: *Super Mario Bros.* Level 1-1 and its text representation.

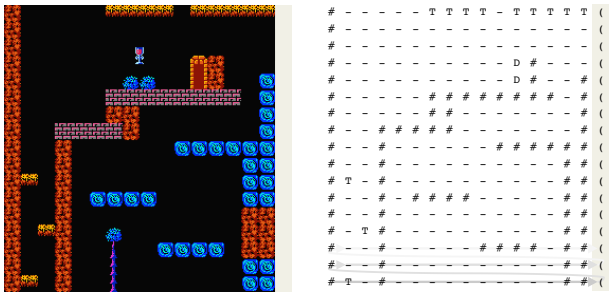

Figure 2: *Kid Icarus* Level 1 and its text representation.

levels of a game, but to games in their entirety. They presented a framework for generating a novel game by using the four space blending process of selecting two input games to blend, creating a generic game concept and then producing a new blended game by combining the generalized elements of the input games. They used the Video Game Description Language (VGDL) (Schaul 2014) to generate a novel game by combining VGDL specifications of *The Legend of Zelda* (Nintendo 1986b) and *Frogger* (Konami 1981).

Dataset

The Video Game Level Corpus. The data for this work is taken from the Video Game Level Corpus (VGLC)[1]. The VGLC (Summerville et al. 2016) is a corpus of levels from a number of games, represented in parseable formats such as *Tile*, *Graph* and *Vector*. It is small compared to traditional ML datasets and the general dearth of sufficient game data for training is a known issue in PCGML. However, the VGLC is an important step towards addressing this issue and has already been used in a sizable amount of games research.

Games. We trained on levels from *Super Mario Bros.* (Nintendo 1985) and *Kid Icarus* (Nintendo 1986a). Both are 2D platformers released for the Nintendo Entertaiment System in the 1980s but differ in that *Super Mario Bros.* levels progress exclusively from left to right where as *Kid Icarus* levels progress from bottom to top. Thus, both are platformers with similar features which may make them compatible for blending, but also have different orientations which

[1] https://github.com/TheVGLC/TheVGLC

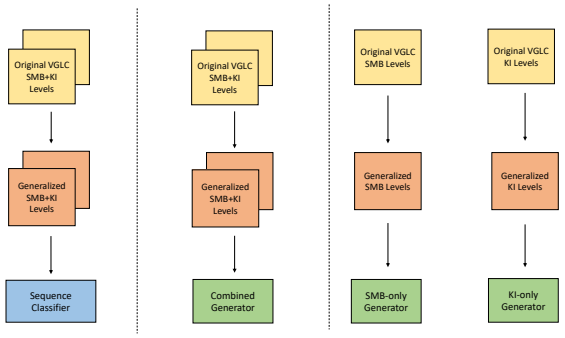

Figure 3: Overview of the training process.

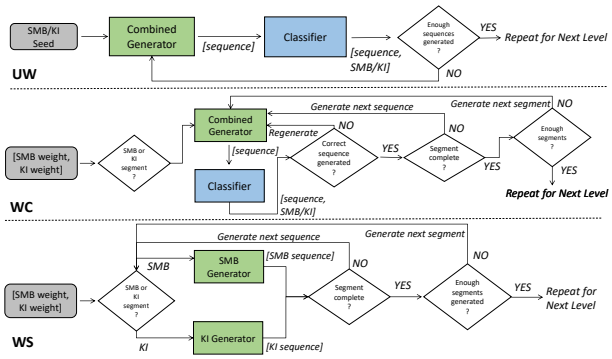

Figure 4: Overview of the generation process. Sequence here refers to either an *SMB* row or a *KI* column. Segments are sets of sequences. Details are given in the Level Generation subsection.

might result in interesting blends. For the rest of the paper, we refer to *Super Mario Bros.* as *SMB* and *Kid Icarus* as *KI*.

Level Representation. The VGLC contains 15 *SMB* levels and 6 *KI* levels, all of which were used for training. Levels are represented in the Tile format using $w \times h$ grids where w is the width and h is the height of the levels. This is stored as a text file with each character mapping to a specific tile. This mapping is stored in a separate JSON file for each game. Parts of example levels and their text representations are depicted for *SMB* and *KI* in Figures 1 and 2 respectively.

Method

This section discusses the different approaches used in this work. High-level overviews of the training and generation phases are given in Figures 3 and 4 respectively.

Blending. Using the conceptual blending framework, our two input spaces are the *SMB* and *KI* level corpora from the VGLC. For the generic space, we mapped the semantically common elements in both games to a uniform tile representation while preserving the tiles that were unique. The only such common elements we identified were solid ground and enemy, and replaced their *KI* representations of # and H in the corpus with the *SMB* equivalents of X and E. The background character ('-') was already the same for both games. Thus, in this generalizing process we use an amalgamation approach as in (Ontañón and Plaza 2010). Prior to training, we converted each level into this new generic representation. These design decisions are matters of personal preference and it is up to the designer to determine the mapping as desired.

LSTM RNNs. Like vanilla neural nets, recurrent neural nets (RNNs) learn weights by backpropagating errors during training. However, unlike standard neural nets where edges are only connected from input layers to hidden layers to output layers, edges in RNNs are also connected from a node to itself over time. Errors are thus backpropagated not just across separate nodes but also over time steps making RNNs suitable for processing sequential input and thus applicable in tasks like handwriting and speech recognition (Graves, Mohammed, and Hinton 2013). However, RNNs suffer from the vanishing gradient problem (Hochreiter 1998) where the error gradient dissipates over time. Hence, standard RNNs are efficient at learning short-term but not long-term dependencies. To address this, Hochreiter and Schmidhuber (1997) proposed the Long Short-Term Memory RNN. LSTMs overcome this problem by adding a memory mechanism to regular RNNs. Being suitable for sequence-based applications, LSTMs are often used for predicting the next item in a sequence given the sequence thus far. This is done by estimating a probability distribution over possible next items and choosing the most likely one as the prediction.

Training on Level Data. For training, we followed the method of Summerville and Mateas (2016). Each level is treated as a collection of sequences, with each individual tile being a point in a sequence. Concretely, a level is a 2D array of characters, as in Figures 1 and 2, with each tile being a character in the array. For *SMB*, we feed in sequences of columns from left to right, since the levels in *SMB* progress horizontally in this direction and for the LSTM to learn the patterns in the level, we need to induce the appropriate ordering for the sequences. Similarly, we feed in *KI* levels in sequences of rows from bottom to top. For uniformity, *SMB* levels were padded with empty space on the top to have columns of height of height 16, while *KI* levels had rows of width 16. This allowed the model to be trained on sequences of 16 character rows or columns, irrespective of the game. To tell the LSTM when one row/column ends and the next begins, we added a delimiter character '(' after every row/column. The LSTM learns via training to generate this character after every 16 characters so that generated levels can be laid out properly.

Due to the disparity in the number of levels for either game, as well as the low total number of levels, we duplicated the Mario and Icarus levels 9 and 21 times respectively, giving us a total of 135 Mario levels and 126 Icarus levels for training. We used a lower number of *KI* levels since its levels were larger than levels in *SMB*. The levels were then split into semi-overlapping sequences of characters. Ultimately, we ended up with a training data set consisting of 149,103 such sequences of *SMB* levels and 149,372 sequences of *KI* levels. To train the LSTM, we used two matrices—the first storing these sequences of characters,

and the second storing the next character in the level for each corresponding sequence. Additionally, the sequences were encoded using One-Hot encoding. In our training models, the LSTM consisted of 2 hidden layers of 128 blocks each. The output layer was sent to a SoftMax activation layer which acted as a categorical probability distribution for the One-Hot encoded tiles. To prevent overfitting, we used a dropout rate of 50%. For all models, we used a learning rate of 0.005, the rmsprop optimizer and categorical cross-entropy loss as the loss function.

Models. We trained 3 different models. One of these used a combined dataset of SMB+KI levels. We also trained a model each on just the SMB levels and on just the KI levels. For training, we used Keras and based code off of the work of Summerville and Mateas (2016), which in turn was based off work by Karpathy (2015). Each model was trained for 50 iterations. While this is a small number, we note that our dataset is much smaller than datasets for traditional ML applications and even with this small number of iterations, we were able to achieve high values for validation accuracy. Training deeper neural nets for a longer period (preferably till convergence of some loss-based criterion) is something to consider for future work. During each iteration of training, 10% of the dataset was held out for validation.

Since SMB and KI levels differ in orientation, an issue in generating levels was determining how to layout generated sequences; i.e. should a generated sequence of 16 characters be laid out like an *SMB* column or a *KI* row? To this end, we trained a classifier on the training corpus and then used it to determine the layout orientation of each generated sequence. For the classifier, we used a sequential model in Keras (Chollet and others 2015) consisting of 1 hidden layer with 256 neurons each in the input and hidden layers and 1 neuron in the output layer, along with a dropout rate of 50%. We used the rectifier activation function on the first 2 layers and a sigmoid activation function on the output layer, along with the rmsprop optimizer and a learning rate of 0.0005. With this network, we achieved a classification accuracy of 91.33% on the *SMB* dataset and 91.29% on the *KI* dataset.

Level Generation. This involves forwarding an initial seed through the trained models multiple times until the desired amount of output is generated. The starting seed is the initial substring of a level string. The trained LSTM repeatedly predicts the next most likely character given the previous characters in the string until enough characters have been predicted to form a level of desired size.

As mentioned before, we created both a regular blended level generator as well as a parametrized variant that affords authorial control. The parameter here refers to weights (between 0 and 1) assigned to each game that determines what percentage of the generated level should be derived from *SMB* and *KI*. Using our 3 training models, we implemented 3 generators—an unweighted generator *UW*, a weighted generator *WC* that used the model trained on the combined corpus of levels and another weighted generator *WS* that used the models trained separately i.e. it consisted of an *SMB*-only sub generator and *KI*-only sub generator that could generate only *SMB* columns and only *KI* rows respectively. *UW* involved sampling from the combined model and using one

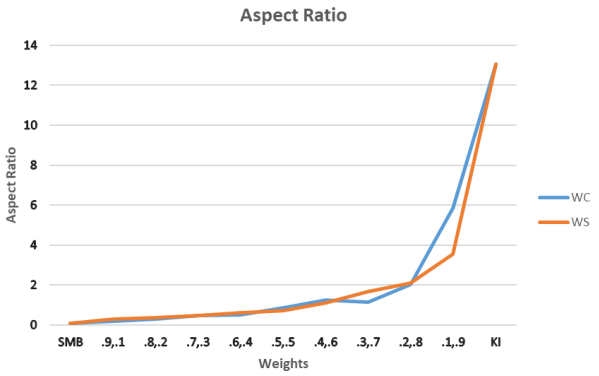

Figure 5: Aspect Ratio

of 2 starting seeds—the initial substring of either an *SMB*-like or a *KI*-like level. For the parametrized generator we had two approaches—either use the combined generator (*WC*) or use the *SMB*-only and *KI*-only generators together (*WS*).

The generation process for *UW* is straightforward. We feed in the starting seed and then iterate until the generator has predicted enough characters to form the entire level, with the classifier used to decide which game a generated sequence belongs to. For the weighted generators, generation took place in segments. Prior to generating each segment, we used the weights to determine if the segment should be SMB-like or KI-like. When using *WS*, depending on if the next segment should be SMB or KI, the *SMB* sub-generator or the *KI* sub-generator was used to generate a fixed-size segment until enough segments had been generated to create the full level. Using this approach, x% of the level segments would be *SMB*-like where as y% would be *KI*-like, where x and y are the pre-specified weights. When using *WC*, the combined generator was used to create sequences of the previously determined game for that segment until it had been fully created, using the classifier to discard any generated sequences that were not for the current segment. For *UW*, we generated levels consisting of 200 sequences. For both *WC* and *WS*, we generated levels consisting of 10 segments, with each segment containing 20 sequences.

Layout. Once generated, the sequences forming the levels are laid out using a basic algorithm. Placing columns after columns and rows after rows is trivial since we just stack them one after another. To place a row after a column, we align its y-coordinate with that of the topmost position in the column on which the player can stand. To place a column after a row, we similarly align the y-coordinate of the top most point on which the player can stand in the column with the y-coordinate of the previous row. The layout function in this work is separate from the generator and thus many different layouts are possible, each necessarily affecting how playable the levels ultimately are. Further investigating the goodness of layouts is important future work.

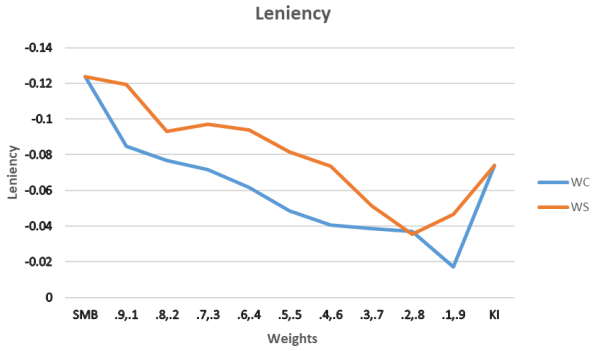

Figure 6: Leniency

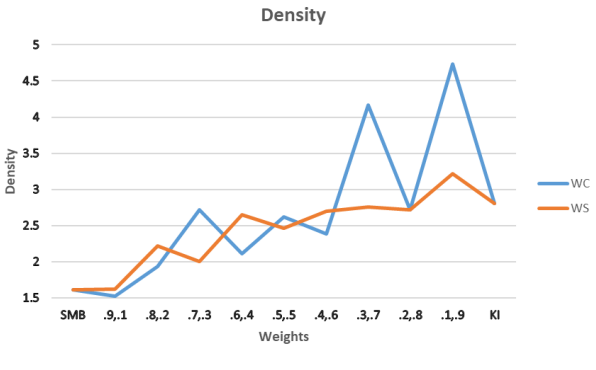

Figure 7: Density

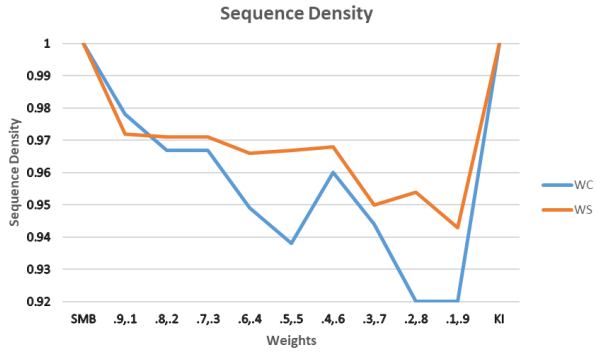

Figure 8: Sequence Density

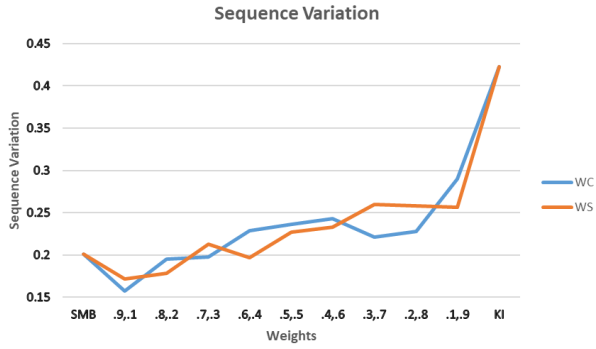

Figure 9: Sequence Variation

Results

Canossa and Smith (2015) and Smith and Whitehead (2010) have proposed several metrics to evaluate the features of generated levels. We used the following in this work.

Leniency. This captures a sense of level difficulty by summing the number of enemy sprites in the level and half the number of empty sequences (i.e. gaps), negating the sum and dividing it by the total number of sequences in the level.

Density. This measures how much of the level can be occupied by the player. For this, we counted the number of ground and platform sprites in the level and divided it by the total number of sequences in the level.

	L	*D*	*SD*	*SV*	*AR*
KI (n=6)	-0.073	2.808	1	0.423	13.073
SMB (n=15)	-0.124	1.61	1	0.201	0.086
UW-SMB	-0.083	1.716	0.929	0.232	0.577
UW-KI	-0.075	1.978	0.925	0.274	0.718
WC (0.5,0.5)	-0.053	3.587	0.938	0.253	0.599
WS (0.5,0.5)	-0.083	2.110	0.964	0.227	0.857

Table 1: Mean values for the metrics for the VGLC levels (top 2 rows) and the generated levels. L, D, SD, SV, AR stand for Leniency, Density, Sequence Density, Sequence Variation and Aspect Ratio respectively.

Sequence Density and Variation. These relate to how different the generated levels are from the training set (Spitaels 2017). Sequence density counts the number of sequences in a level that also exist in the training set and then divides it by the total number of sequences in the level. Sequence variation is similar but counts each occurrence of a training sequence once in a generated level.

Aspect Ratio. This is calculated by dividing the number of rows (height) in a level by the number of columns (width).

We used the unweighted generator *UW* (with both *SMB* and *KI* seeds) and weighted generators *WC* and *WS* with weights of 0.5 to create 100 blended levels each. Results for the above metrics are given in Table 1.

Expressive Range. Additionally, we wanted to look at the expressive range of the weighted generators as their weights are varied. The expressive range of a generator (Smith and Whitehead 2010) is the style and variety of the levels it can generate. To visualize the range of the weighted generators and compare them with each other, we generated 10 levels for each of 10 pairs of weights. Figures 5, 6, 7, 8 and 9 show the range of the weighted generators as the weights are varied between the two games. Example generated levels for unweighted and weighted generators are shown in Figures 10 and 11 respectively.

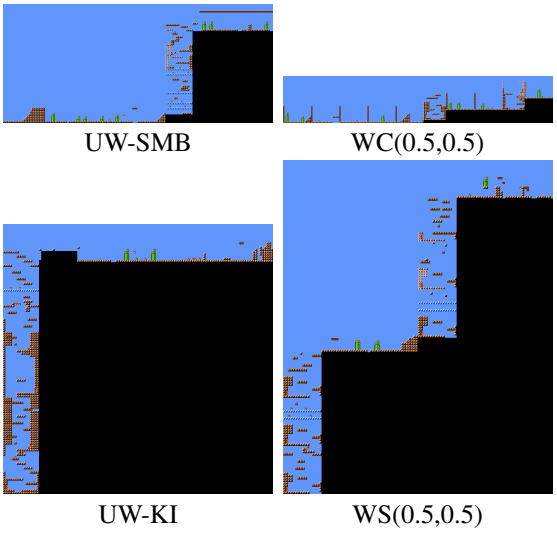

Figure 10: Example levels made by generators in Table 1

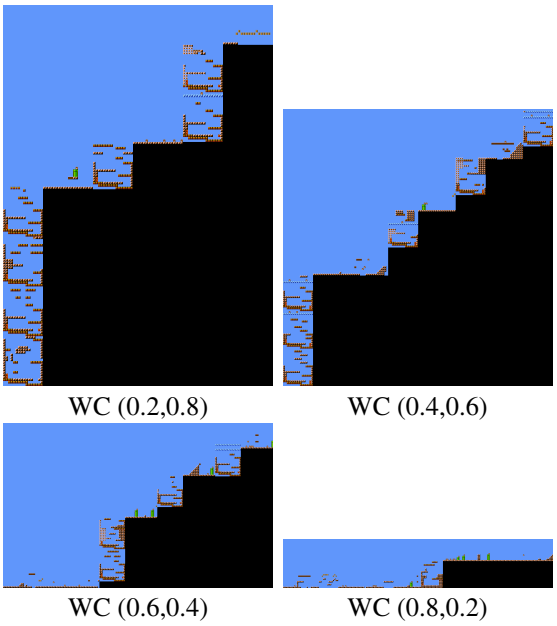

Figure 11: Example levels made by generator WC

Discussion

Figures 5, 6, 7, 8 and 9 suggest that altering the weights does impact the type of levels generated and allows the designer to roughly interpolate between *SMB* and *KI*. For both *Aspect Ratio* and *Sequence Variation*, as we move from a weight pair of (high-*SMB*, low-*KI*) to (low-*SMB*, high-*KI*), the values move from being closer to the *SMB* corpus to being closer to the *KI* corpus, as expected. This is also mostly true for *Leniency* and *Density* though interestingly for these, while the values follow the same trend, the blends with higher amount of *KI* seem to have higher values than the *KI* corpus itself. That is, *KI*-heavy generated levels were more lenient and dense than the actual *KI* levels used in training where as the *SMB*-heavy generated levels were closer to the originals in terms of *Leniency* and *Density*. For *Sequence Density*, the *SMB* and *KI* corpora have values of 1 by definition. It is interesting however that there is a slight decrease in this value as the amount of *KI* is increased in the blend. As for comparing *WC* and *WS*, *WS* seems to adhere more closely to the range of values between those of the *SMB* and *KI* corpora while *WC* generates more novel sequences as evidenced by the lower values for *Sequence Density*. Overall, these results suggest that by using the methodology outlined in the paper, it is possible to generate levels that are a mix of levels from two other games but that can also be made to be more like one than the other, as desired by the designer. Moreover, the unexpected deviations highlighted above suggest that in addition to emulating training data and capturing its inherent properties, these generative methods can also produce novelty as evidenced by some blended levels having properties outside of the expected range within the two input games. In the future, more thorough and rigorous approaches combined with richer datasets might lead to generative models and processes that are both more controllable as well as capable of producing more interesting and novel results.

Conclusion and Future Work

In this work, we trained LSTMs on levels from *Super Mario Bros.* and *Kid Icarus* to generate levels that were blends of the levels from these two games. This suggests that leveraging PCGML may help realize the VGDL-based game generation framework proposed by Gow and Corneli (2015).

There are several directions for future work. An obvious limitation of this work is that no playability tests were run on the generated levels nor any playability or path-based information used in training. Thus, levels are currently not traversable from start to finish using either *SMB* or *KI* mechanics. Future work could involve using an agent to carve out a path post-generation or encoding path information into the training corpus as in Summerville and Mateas (2016).

Having said that, the lack of playability is not surprising since we would expect blended levels to require blended mechanics to be playable. While levels are integral to a game, so too are the mechanics and player-world interactions. Blending two games thus necessitates blending their mechanics as well. Gow and Corneli (2015) demonstrate this in their VGDL example by combining the mechanics of *The Legend of Zelda* with *Frogger*. The feasibility of applying the technique discussed in our work towards game mechanics is worth looking into for future work. It might additionally be possible to leverage evolutionary algorithms to evolve game mechanics that are compatible with the newly blended game and are based off of the mechanics of the original games being blended.

References

A.I. Design. 1980. Rogue.

Bou, F.; Corneli, J.; Gomez-Ramirez, D.; Maclean, E.; Pease, A.; Schorlemmer, M.; and Smaill, A. 2015. The role

of blending in mathematical invention. In *Proceedings of the 6th International Conference on Computational Creativity*.

Braben, D., and Bell, I. 1984. Elite.

Broderbund. 1983. Lode Runner.

Cambouropoulos, E.; Kaliakatsos-Papakostas, M.; and Tsougras, C. 2014. An idiom-independent representation of chords for computational music analysis and generation. In *Proceedings of the joint 11th Sound and Music Computing Conference (SMC) and 40th International Computer Music Conference (ICMC)*.

Canossa, A., and Smith, G. 2015. Towards a procedural evaluation technique: Metrics for level design. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.

Chollet, F., et al. 2015. Keras. https://keras.io.

Dahlskog, S.; Togelius, J.; and Nelson, M. 2014. Linear levels through N-grams. In *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, 200–206.

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

Fauconnier, G., and Turner, M. 2008. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*. Basic Books.

Goguen, J. 1999. An introduction to algebraic semiotics with application to user interface design. *Computation for Metaphors, Analogy and Agents* 242–291.

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*.

Graves, A.; Mohammed, A.-R.; and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (icassp)*, 6645–6649.

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

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

Guzdial, M., and Riedl, M. 2017. Combinatorial creativity for procedural content generation via machine learning. In *Proceedings of the First Knowledge Extraction from Games Workshop*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Hochreiter, S. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6(2):107–116.

Hoover, A.; Togelius, J.; and Yannakakis, G. 2015. Composing video game levels with music metaphors through functional scaffolding. In *First Computational Creativity and Games Workshop*.

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

Karpathy, A. 2015. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Konami. 1981. Frogger.

Nintendo. 1985. Super Mario Bros.

Nintendo. 1986a. Kid Icarus.

Nintendo. 1986b. The Legend of Zelda.

Ontañón, S., and Plaza, E. 2010. Amalgams: A formal approach for combining multiple case solutions. In *International Conference on Case-Based Reasoning*.

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

Schorlemmer, M.; Smaill, A.; Kuhnberger, K.-U.; Kutz, O.; Colton, S.; Cambouropoulos, E.; and Pease, A. 2014. Coinvent: Towards a computational concept invention theory. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Shaker, N.; Togelius, J.; and Nelson, M. 2016. *Procedural Content Generation in Games*. Springer International Publishing.

Smith, A., and Mateas, M. 2011. Answer set programming for procedural content generation. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):187–200.

Smith, G., and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*.

Smith, G.; Whitehead, J.; and Mateas, M. 2011. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):201–215.

Snodgrass, S., and Ontañón, S. 2014. Experiments in map generation using Markov chains. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*.

Snodgrass, S., and Ontañón, S. 2015. A hierarchical MdMC approach to 2D video game map generation. In *Proceedings of the Eleventh Artificial Intelligence and Interactive Digital Conference*.

Snodgrass, S., and Ontañón, S. 2016. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*.

Snodgrass, S., and Ontañón, S. 2017. An approach to domain transfer in procedural content generation of two-dimensional videogame levels. In *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)*.

Spitaels, J. 2017. MarioNet: Generating realistic game levels through deep learning. Master's Dissertation.

Stanley, K., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10(2):99–127.

Summerville, A., and Mateas, M. 2015. Sampling Hyrule: Sampling probabilistic machine learning for level generation. In *Proceedings of the Eleventh Artificial Intelligence and Interactive Digital Conference*.

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

Summerville, A.; Behrooz, M.; Mateas, M.; and Jhala, A. 2015. The learning of Zelda: Data-driven learning of level topology. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The Video Game Level Corpus. *Proceedings of the 7th Workshop on Procedural Content Generation*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A.; Isaksen, A.; Nealen, A.; and Togelius, J. 2017. Procedural content generation via machine learning (PCGML). In *Foundations of Digital Games Conference*.

Togelius, J.; Yannakakis, G.; Stanley, K.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):172–186.

Yannakakis, G.; Liapis, A.; and Alexopoulos, C. 2014. Mixed-initiative co-creativity. In *Foundations of Digital Games Conference*.