Controllable and Coherent Level Generation: A Two-Pronged Approach

Justin Mott, Saujas Nandi, Luke Zeller
University of North Carolina at Chapel Hill
justinmmott@gmail.com, saujas.nandi@gmail.com, rlukezeller@gmail.com

Abstract

Procedurally generating high-quality content is an important problem in the design and development of video games. A key objective of procedural level generation is *controllability*: the ability to generate novel levels with a predetermined goal, such as maximizing difficulty. Previous work has achieved this for platformer games using deep-learning and evolutionary methods, but such methods often lead to invalid or asymmetric structures within the generated levels. On the other hand, generative approaches using LSTMs and Markov chains have succeeded in preserving local coherence but suffer from limited controllability. In this paper, we introduce a framework for generating controllable and locally coherent levels in platformer games. Our framework extends a prior GAN and evolutionary strategy-based approach with LSTM Recurrent Neural Networks and hyperparameter optimization. We use sequence transformation via a LSTM to improve aesthetic coherency, while tuning fitness function hyperparameters for specific objectives and measuring overall level quality through human evaluation. To verify our approach, we create levels for *Super Mario Bros* intended to be novel and challenging to expert players. We demonstrate that the framework is able to successfully generate such levels in a controllable manner, while also resolving invalid and unaesthetic local structures.

Introduction

Efficiently creating high-quality levels is a key challenge in the video game industry. Video game development can take thousands of man-hours to ensure that the player's experience is seamless and that the player may immerse themselves in the video game world. In today's video game market, where annual releases of video game are common, developers face significant time constraints that can limit their ability to build creative and complex levels. As a result, Procedural Content Generation (PCG), which enables the generation of levels with limited human involvement (Goodfellow et al. 2014a), has gained notable attention in both academic and professional spheres (Shaker, Togelius, and Nelson 2016).

Traditionally, PCG has required the involvement of domain experts when designing generators for a specific video game (Volkovas et al. 2019). The compositional methods and the metaheuristic search algorithms described in (Togelius, Justinussen, and Hartzen 2012) and (Togelius et al. 2011) are examples of strategies that require expert knowledge to design the parameter space that these methods will search over. It is difficult for businesses to justify exploring these techniques, as they are specific to a title and still take many man-hours to develop. However, PCG Machine Learning (PCGML), which involves applying machine learning techniques to PCG, has been shown to significantly reduce the amount of time spent designing game-specific systems (Summerville et al. 2018). As a result, PCGML techniques can lead to the development of automated game design systems which are generalizable and can be reapplied to a variety of similar games. Unfortunately, applications of PCGML methods suffer from a lack of *controllability*, the ability to generate novel levels with a prescribed goal, and *coherence*, the structural and aesthetic quality of a level. Due to their generalizability, PCGML techniques and frameworks are often difficult to configure for meaningful goals or structures within a given game. Achieving coherence is difficult because ML methods are inherently approximate and often noisy, preferring solutions that live near a global maximum to precise local maxima. For this reason, PCG with such techniques will sometimes result in asymmetric and unaesthetic local structures that appear out of place.

In this paper we present a framework to address the aforementioned issues facing current PCGML methods. Considered generically, our system assists with the procedural generation of aesthetically coherent levels which can be controlled for specific, predefined goals such as maximizing difficulty or number of solution paths. We accomplish this using a two-pronged approach: controllability is achieved with a fitness function and human evaluation of level quality, and coherence is reintroduced through the employment of a Long Short Term Memory Recurrent Neural Network (LSTM) (Hochreiter and Schmidhuber 1997). Our paper focuses on Super Mario Bros (SMB1), which has been the subject of PCGML research for several years; in our work, we aimed to create levels that would prove to be challenging,

even for expert level players.

Background

As machine learning techniques are constantly improving, a portion of these advancements have propagated throughout PCG research (Smith 2014). Over the last few years, there has been a broad effort in the PCG research community to develop new PCGML techniques; from GANs (Park et al. 2019) to LSTMs (Summerville and Mateas 2016) to co-creative approaches (Guzdial, Liao, and Riedl 2018). Our work extends and generalizes techniques such as these; in this section, we present an overview of the background relevant to our contributions.

Controllability

As previously mentioned, we define controllability as the ability to generate novel levels with prescribed goals. Controllability is often recognized as a key feature of PCG techniques and frameworks, and has been the subject of much prior work. To improve controllability our framework builds upon the PCGML methods used by (Volz et al. 2018), who successfully employed Generative Adversarial Networks (GANs) (Goodfellow et al. 2014b) to generate novel levels and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen, Müller, and Koumoutsakos 2003) to control features of generated levels.

GANs GANs are a pair of convolutional neural networks, consisting of a generator and a discriminator, which together compete in an adversarial manner. During the training process, the generator attempts to create data representative of a given training set, while the discriminator attempts to distinguish this input from real training data. As mentioned above, Volz et al. employed GANs to assist with Super Mario Bros. content generation. During training, they had the generator produces playable SMB1 levels while the discriminator was trained on these generated levels along with segments from SMB1 level 1-1. In their implementation, they utilized a pytorch model of a Wasserstein GAN (Arjovsky, Chintala, and Bottou 2017). WGANs are a variation on Deep Convolutional GANs (Radford, Metz, and Chintala 2015) that effectively minimize the difference between the model's distribution and the actual distribution. They then used the results of the GAN in the next stage of their level generation, the CMA-ES evolution process.

CMA-ES CMA-ES is an evolutionary strategy that operates on *latent vectors*, which are hidden (i.e. latent) representations of the data under consideration. As in Volz et al. the latent vectors serve as input to the generator network, which transforms them into a level for which fitness can be calculated. The CMA algorithm takes advantage of statistical properties of these vectors to generate and mutate populations while optimizing for a given fitness function by identifying clusters of latent vectors that have the desired properties. Using the fitness values, the algorithm creates a new generation by combining the highest fitness examples from the current generation with carefully constructed but non-deterministic mutations. Although the evolution process introduces several random mutations in each level, many of these mutations could have no effect on the fitness function; for example, adding blocks that cannot be reached as seen in Figure 1, or deleting tiles contained in staircases as seen in Figure 2. Despite many mutations resulting in no practical difference in level fitness, CMA-ES has proven to be effective in improving the population fitness over time.

In order to generate a new population, the fitness of every level in the current population must be calculated. Volz et al. used an AI agent to complete levels and calculated their fitness from metrics recorded during the AI's execution. The AI agent used was the A* agent developed by Robin Baumgarten for the 2009 Mario AI competition (Togelius, Karakovskiy, and Baumgarten 2010). Their primary metric for fitness is derived directly from the number of jumps performed by the AI agent in completing a level. For our purposes, this approach is limited in its capacity to identify specific features within a level and thus lacks sufficient controllability.

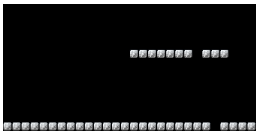

Figure 1: Unreachable Blocks

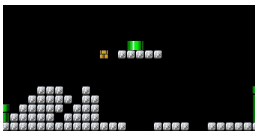

Figure 2: Staircase with Gaps

Level Representation Our initial level generation process (consisting of GAN level generation and CMA-ES tuning/optimization) is based closely on the workflows used by Volz et al. in their previously mentioned PCG work. We describe this workflow here. To begin, Mario levels are collected from the Video Game Level Corpus (Summerville et al. 2016) which contains a collection of text-based representations of such levels. The tiles are represented with ASCII characters and converted into integers as shown in Table 1. The integers are then one-hot encoded before being passed into the WGAN for training and testing. During training and generation, levels are divided into segments of fixed and predetermined length and height. Several such segments can be merged to produce levels of the same length as those in the original game.

Tile	ASCII	ID
Ground	X	0
Breakable	S	1
Air	-	2
Question Block	?	3
Empty Question Block	Q	4
Goomba	E	5
Top-left Pipe	<	6
Top-right Pipe	>	7
Left Pipe	[	8
Right Pipe	]	9

Table 1: Encoding of Mario Levels

After training the WGAN, level generation may begin. In this procedure, randomly selected latent vectors are passed

to the WGAN generator which returns level segments of a predetermined size. These segments are then used as an initial population for the CMA-ES evolution process. The evolution algorithm then proceeds in a standard manner, running its evolutionary strategy according to a given fitness function; this fitness function will take each new population of latent vectors and pass them back to the generator network, then performs an evaluation on the resulting level. After multiple iterations of this process, the single latent vector with the best fitness is returned and saved. Note that each latent vector corresponds only to a small segment of a full-length level; we can then repeat this process as many times as necessary and stitch together the resulting segments to create a full level.

Coherence

While strong controllability is arguably the most important factor in determining the overall utility of a PCG framework, local coherence in the generated levels is an integral aspect of practical PCG. Locally, generated levels should be structured in a manner consistent with real levels and should possess a similar amount of symmetry (or lack thereof). In particular, any visual properties or constraints held by all human-authored levels should also be held (or approximated as well as possible) by any generated level. Examples where this is relevant in SMB include restrictions to valid pipe structures and proper ground tile/stair placement and shape.

LSTMs LSTMs are a type of Recurrent Neural Network comprised of carefully-designed memory blocks. Within each block is a memory cell with self-connections storing its temporal state as well as its gates. The gates are multiplicative units that control the flow of information. Within each memory cell there is an input gate, a forget gate, and an output gate. The input and output gates moderate how a given activation will affect the memory cell as well as the network as a whole, while the forget gate moderates the flow of the internal state of the cell into its own input (Sak, Senior, and Beaufays 2014). LSTMs are effective at learning local and semi-local structures and patterns apparent within training data (Karpathy, Johnson, and Fei-Fei 2015) and have been used to generate new levels by training on existing levels from the VGLC (Summerville and Mateas 2016). This method of PCGML by Summerville et al. achieves good local coherence, but has limited controllability. The LSTM is able to accurately reproduce structures found within the training data, creating objects such as stairs and pipes correctly. However, because it is trained on levels from the VGLC and not parameterized in a meaningful way, controlling the output requires the creation of more tailored training data. This undermines the utility of PCG, because it requires additional levels to be hand-crafted in order to add additional features.

Approach

We build on the WGAN and CMA-ES based approach of Volz et al. in three key ways. First, we introduce a novel fitness function for CMA-ES, which is well correlated with level difficulty. Next, we describe and carry out a complementary human-evaluation process in order to optimize the hyperparameters in our fitness function without requiring or assuming knowledge of the game being played. Lastly, we use a LSTM to improve the coherency of the generated level. We will describe our approach by covering the controllability and coherency components of our framework in two parts. For a more general overview of our framework refer to Figure 3.

Controllability

Frame Windows In order to optimize for challenging levels, we needed to identify a metric that corresponded to a level's difficulty. We utilized a measure of difficulty based on the number of frames that an action is feasible for a player to perform, referred to as a frame window, where a frame is one video frame of the game. We determine the frame window for each action via the following process: First we record the actions an AI agent makes in order to complete a level. Next, we perturb the beginning of each action, one frame at a time, and check if the level is still completable with this modified set of actions. The frame window of an action is the number of perturbations that still results in the level being completed. Actions with small frame windows are more difficult for human players to execute successfully than actions with larger frame windows, as smaller frame windows force players to react more quickly in order to pass the level. In SMB1, pressing the jump key is the main action that players use, as most levels can be completed without changing directions. For our implementation, we shifted one jump at a time without modifying the remaining jumps. For each jump j, we calculated the number of frames F_j that the start of j could be shifted to, while still completing the level. We defined the difficulty of a level that was completed by an AI agent with a set of jumps J as:

$$D = \frac{1}{\sum_{j \in J} F_j}$$

Fitness Function When creating our fitness function, we needed to take two separate components of difficulty into account, the local difficulty of specific actions, and the global difficulty of completing a level as a whole. The notion of difficulty described above aggregates the local difficulty of a level, but fails to account for the global difficulty. To compensate for this, we also used a global measure of level difficulty based on the fraction of times that an AI agent was successfully able to complete the level. The A* agent used in our implementation was non-deterministic, and in practice was able to complete difficult levels less than 30% of the time and easy ones 80% or more of the time. As a result, we were able to run this A* agent on a level multiple times and treat the fraction of successful runs as a measure of the global difficulty of the level. However, if an AI agent with high variance is not available for some game, one can substitute multiple AI agents with varying degrees of ability to achieve the same result by using the fraction of agents that successfully complete a level as a measure of the global difficulty.

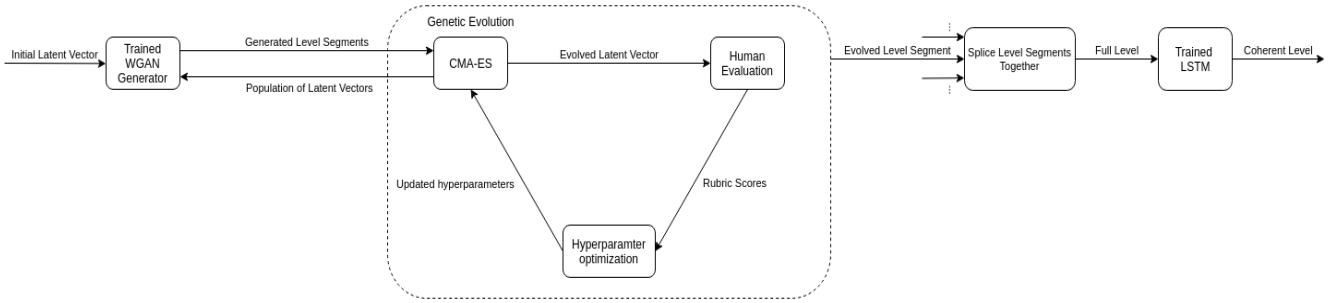

Figure 3: Level Generation Pipeline

One problem with attempting to maximize the difficulty of generated levels is that infeasible levels (i.e. those which are impossible to complete), despite being the most difficult levels, are typically unwanted as they are not interesting to play and would lead to player frustration. To account for this, we penalize such levels by assigning them a negative fitness value corresponding to the length of the level that can be completed. More formally, let f be the maximum fraction of the level that any AI agent has traversed. If $f < 1$, which typically only happens when the level cannot be completed, the level is assigned a fitness of $f - 1$, which rewards levels that have a larger playable area while also ensuring that incompletable levels are always less fit than completable levels.

We used hyperparameters, $k_1 \dots k_3 \in \mathbf{R}$ as weights for the various components of our fitness function. The fitness value F of a level L with an aggregated local difficulty of ℓ and a global difficulty of g, and where f is the greatest fraction of the level that was completed by any AI agent:

$$F = \begin{cases} k_1 \cdot \ell + k_2 \cdot g & \text{if} f = 1.0 \\ k_3 \cdot (f - 1) & \text{if} f < 1.0 \end{cases}$$

Because the CMA-ES implementation we used was designed to find the minima of the given fitness function, in practice we negate the result of F when calculating fitness.

Human Evaluation To find the optimal values of the hyperparameters mentioned above, we used a fixed step-size random search on the 3D hyperparameter search space, based on the algorithm introduced by (Rastrigin 1963). To evaluate a hyperparameter $H = (x, y, z)$, we first created a fitness function by fixing the hyperparameters for the function F with $k_1 = x$, $k_2 = y$, $k_3 = z$. We then used GAN generation and CMA-ES with this fitness function to generate a challenging level with the properties determined by the specific hyperparameter values. At this point, the generated level was played by a human evaluator with prior expert-level speedrunning experience. The evaluator rated each level on the enjoyability of the playing experience; this rating was then used as the fitness for H. We chose to restrict the ratings to lie between 1 and 15 inclusive, but this restriction is a practical matter and does not affect the results of the random search. For clarity and consistency, we interpreted the following intervals as follows:

[1, 5):	Boring or unplayable level Would not play again
[5, 10):	Decent level with some good qualities Would play again
[10, 15]:	Excellent level with many highlights Would play repeatedly

Table 2: Human Evaluation Rating Interpretation

The random search process is as follows for a predetermined step-size d and a population size p: to initialize the random search, we generate a population of p random hyperparameters and evaluate them. Now for each iteration of the random search, we choose the best hyperparameter in our population as our candidate solution, and randomly generate $p - 1$ new hyperparameters that are distance d from this candidate. These $p - 1$ new hyperparameters form the new population along with the candidate solution. After human evaluating the new hyperparameters, we repeat the process until we generate levels that are consistently interesting. Augmenting our CMA-ES evolution with human evaluation provides a systematic way to create challenging yet fun levels without requiring significant prior knowledge about any particular game.

Coherence

As discussed earlier, the CMA-ES process will sporadically create incoherent local structures, as exemplified by Figures 7 and 8. We chose to use a LSTM to help repair these local inconsistencies and improve the level's coherency. While local Markov-chain methods have demonstrated success at replicating pipes and other local structures, they fail to effectively capture the semi-local aesthetic structure of stairs as shown in (Snodgrass and Ontan 2017) and (Snodgrass and Ontañón 2014). On the other hand, LSTMs have been shown to successfully maintain local and semi-local coherence for level generation (Summerville and Mateas 2016). The LSTM that we use was implemented with pytorch and has hidden states with 256 dimensions and an embedding dimension size of 256.

We created our training data by perturbing the levels given by the VGLC. We narrowed down the levels in the VGLC to 16 levels by choosing all of the above-ground SMB1 levels, as opposed to sky and underground levels, since the GAN

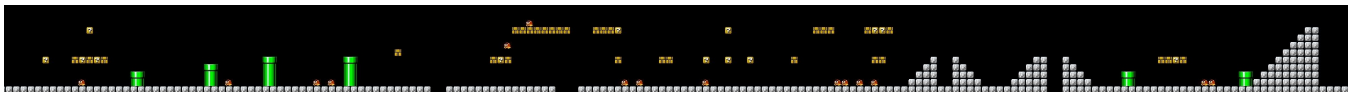

Figure 4: Level 1-1

Figure 5: Level 1-1 Perturbed

Figure 6: Level 1-1 Fixed

was trained on an above-ground level and there are vast differences between the three level types. By perturbing this set of 16 levels and training the LSTM to convert the perturbed levels back to the original, we were able to teach the LSTM to fix many of the common sources of local incoherencies: broken pipes, misaligned staircases, and random ground tiles. The perturbations were done on a random subset of the following tile types: pipe tiles were replaced with other tile types, staircase tiles were either replaced by an empty tile or shifted horizontally, and air tiles were replaced with ground tiles. These perturbations can be found by comparing Figures 4 and 5. Many of the pipes have been altered in some way, with only 1 pipe left untouched, while ground tiles have been added throughout the level and the staircases more closely resemble a bridge.

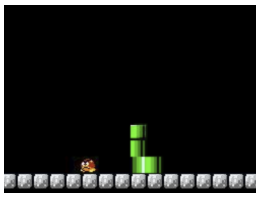

Figure 7: Broken Pipe

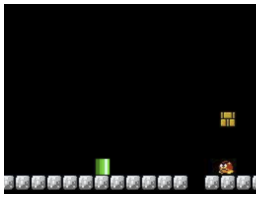

Figure 8: Random Pipe Tile

Controllability

With the training data created, we then flatten each level into a sequence by iterating through its columns. We chose this representation so that adjacent tiles are separated by at most 15 tiles, rather than the 200 tile spacing we would get if we had used a row-based representation. We then train the LSTM on the sequence-representations of the original and perturbed levels.

Results

We stored the human evaluation rating of the candidate solution in each iteration of the hyperparameter optimization stage of our framework, with the results shown in Figure 10.

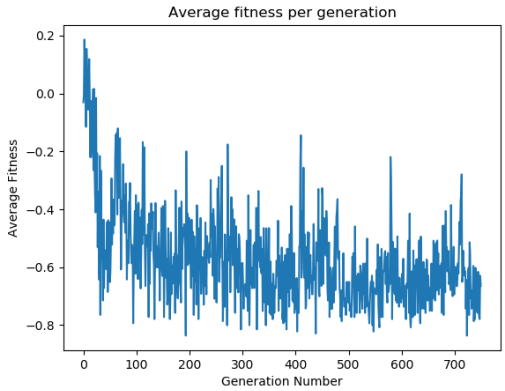

Figure 9: Average population fitness per CMA-ES Iteration. Lower values are better.

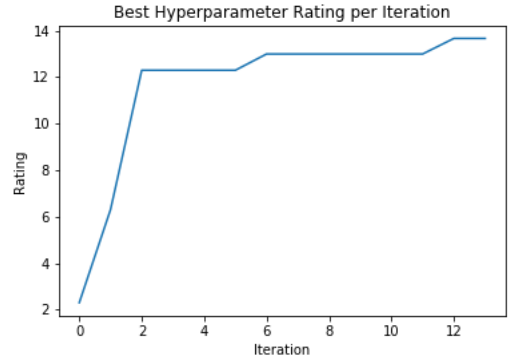

Figure 10: Best Level Rating for each Iteration of the Hyperparameter Optimization Random Search

The random search proved to be effective as we were able to find hyperparameters that earned a rating of 13 points out of 15 in a small number of iterations. For these results, we used a population size of $p = 3$ and a step size of $d = 5$. In Figure

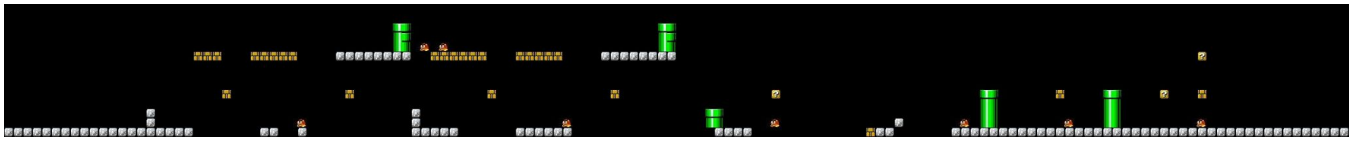

Figure 11: Generated Level

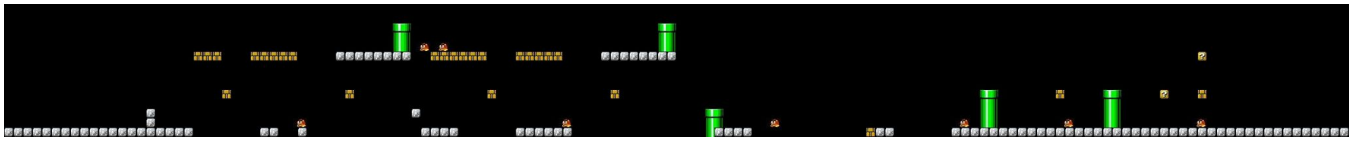

Figure 12: Generated Level Fixed

11, we can see an example of a challenging generated level. This level is completable but still requires several difficult jumps, including a jump that involves landing on exactly one tile, and another that requires the player to jump nine tiles (the maximum possible jump length is ten tiles). However, this generated level has some aesthetic flaws, as can be seen in the incorrect pipes and random floating blocks. These problems are mostly fixed by the LSTM.

Coherence

After training the LSTM, we generated a testing data set using the same types of perturbations as when creating the training data set, and achieved a testing accuracy of 99.05%. We find this accuracy by running the LSTM on the testing data set and then calculating the following

$$\text{Test Accuracy} = \frac{\text{\# of Tiles correct}}{\text{Total \# of Tiles}}$$

The LSTM's performance can be observed by comparing Figures 4, 5, and 6. It is clear that the LSTM has modified the perturbed level and its result is not far from the original level. All the blocks have been fixed and the staircase at the end now closely resembles the original except for the block at the far right. The staircases in the middle were not completely fixed, but they are significantly closer to the original level than what is given in the perturbed level.

To check the LSTMs effectiveness in our pipeline, we also observed how it performed on levels that were generated by the GAN + CMA-ES process, such as the level shown in Figure 11. When compared to the LSTM-processed level, shown in Figure 12, we can see that the LSTM was able to fix multiple pipes and some inconsistencies in the ground tiles. Crucially, locations without inconsistencies, such as the isolated breakable tiles, are left unchanged, which is key to preserving the completability and difficulty of the level.

Limitations and Future Work

Given the success of autoencoders in repair/denoising applications (Jain et al. 2016), we would like to experiment with them, as well as Seq2Seq methods (Vaswani et al. 2017), as alternatives to LSTMs for level repair. This will allow the model to make more informed decisions about a particular tile as it will have information about surrounding (and particularly future) tiles. Given more time, we would also like to take levels generated by the CMA-ES and fix them by hand for the purpose of training the model on completely representative data. Furthermore, we found that *Infinite Mario Bros.* has non-deterministic behavior in its physics engine; this affects the results of our fitness function calculation, as the feasibility of a perturbed action set may be incorrectly judged. The work by Volz et al. and (Snodgrass and Ontañón 2016) also explore level representation-based objectives, in which a level must meet various criteria (e.g. a specific number of enemies or gaps), the values of which are determined via prior calculation. These target values, effectively additional hyperparameters, could instead be chosen via human evaluation, which has the benefit of being reproducible as well as customizable for the target audience; the ideal hyperparameters for an expert level player could vastly differ from those of a beginner. Lastly, the human evaluation of hyperparameters could be distributed across multiple players to avoid overfitting a single person's preferences as well as reducing the overall bias and amount of work necessary per-person to achieve usable results.

Conclusion

We have presented a framework that allows for the creation of a PCG model enabling controllability via a generic fitness function and human evaluation, while also maintaining coherence via LSTMs. We believe our framework can be naturally generalized to other similar AI-solvable platformer games, and with appropriate modifications, that it may be further generalizable to many games solvable by an AI agent, so long as levels can be naturally represented as text and linearized. While our frame-window based approach might not extend naturally to games in other genres, the success-ratio metric for global difficulty is more broadly applicable. In any case, different fitness functions can be defined based on the individual characteristics of the game under consideration.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 214–223.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014a. Generative adversarial nets. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 2672–2680.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014b. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Guzdial, M.; Liao, N.; and Riedl, M. 2018. Co-creative level design via machine learning. *arXiv preprint arXiv:1809.09420*.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation* 11(1):1–18.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9:1735–80.

Jain, R.; Isaksen, A.; Holmgård, C.; and Togelius, J. 2016. Autoencoders for level generation, repair, and recognition.

Karpathy, A.; Johnson, J.; and Fei-Fei, L. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Park, K.; Mott, B.; Min, W.; Boyer, K.; Wiebe, E.; and Lester, J. 2019. Generating educational game levels with multistep deep convolutional generative adversarial networks. In *Proceedings of the IEEE Conference on Games*.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Rastrigin, L. A. 1963. The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control* 24(10):13371342.

Sak, H.; Senior, A.; and Beaufays, F. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural content generation in games*. Springer.

Smith, G. 2014. Understanding procedural content generation: a design-centric analysis of the role of pcg in games. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, 917–926. ACM.

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, 780–786. AAAI Press.

Snodgrass, S., and Ontañón, S. 2014. Experiments in map generation using markov chains.

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

Summerville, A., and Mateas, M. 2016. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930*.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Villar, S. O. 2016. The VGLC: the video game level corpus. *CoRR* abs/1606.07487.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games* 10(3):257–270.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; 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.

Togelius, J.; Justinussen, T.; and Hartzen, A. 2012. Compositional procedural content generation. In *Proceedings of the The third workshop on Procedural Content Generation in Games*, 16. ACM.

Togelius, J.; Karakovskiy, S.; and Baumgarten, R. 2010. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, 1–8. IEEE.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Volkovas, R.; Fairbank, M.; Woodward, J.; and Lucas, S. 2019. Mek: Mechanics prototyping tool for 2d tile-based turn-based deterministic games. *arXiv preprint arXiv:1904.03540*.

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A. M.; 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 (GECCO 2018)*. New York, NY, USA: ACM.