The Learning of Zelda: Data-Driven Learning of Level Topology

Adam J. Summerville Morteza Behrooz Michael Mateas Arnav Jhala
University of California, Santa Cruz
1156 High Street
Santa Cruz, California, USA
{asummerv,morteza}@ucsc.edu {michaelm,jhala}@soe.ucsc.edu

ABSTRACT

The majority of Procedural Content Generation (PCG) research has made use of human authored rules, heuristics and evaluation metrics. Machine learning techniques have gone relatively unused in PCG. We introduce a data-driven level generation approach, and apply it to the of dungeons for Zelda-like Action Roleplaying Games (ARPGs). We use Bayesian Networks (BNs) to learn distributional information about level topology. The learned networks can then be sampled to generate levels that have the same statistical properties as human authored levels.

Keywords

procedural content generation, machine learning, level design, data-driven level design, games, probabilistic learning, Bayesian network

1. INTRODUCTION

Procedural Content Generation (PCG) is the algorithmic creation of new artifacts encompassing a wide set of techniques. Despite decades of craft practice, PCG research is a nascent field. Generating levels for video games has been one of the most active areas of PCG research. Levels provide the primary space for players to interact with the game, and the design of levels is interconnected with the atomic content of the game as well as the underlying mechanics.

Machine learning in the context of PCG has only just begun to be explored in the last few years. Most PCG techniques have been based off of a designer encoding some system of rules or evaluation functions that determine the fitness of a generated level. Learning directly from levels allows a designer to generate new levels without needing to formalize their design decisions. Instead they need only find, or create, representative levels and allow the machine to learn the how to design levels.

Action Role Playing Games (ARPGs), with Legend of Zelda being the exemplar, present an interesting challenge for PCG. The machine learned PCG techniques above have only been applied to platformers that progress from left to right. In ARPGs the levels have complex topology that the player must explore. For the sake of this paper we will discuss the levels at three levels of abstraction. The highest is the *mission space* of the level. This is composed of the tasks the player must accomplish to complete the level, e.g., collecting a key to open a door. At the lowest level of abstraction there is the *physical space* of the level. Typically, ARPG levels are composed of a number of discrete rooms connected by doors and this is the finest level of granularity that most research has focused on [11] [7] [12] [19], although there have been a few approaches that have looked at the placement of atomic game components [9] [18]. The *player space* binds the *mission space* and *physical space*. While completing the *mission space*, the player explores the *physical space* of the level. Often, the player must backtrack through previously explored areas of the *physical space*. It is the path that players take that makes up the *player space* of the level.

There are many machine learning techniques that might be applied to learning levels. Previous work has used Markov chain generation [6] and non-negative matrix factorization [16]. However, it is not enough to learn from existing levels. We want a machine learning technique that is parameterizable over common design decisions (length of level, difficulty, etc.), that allows for artifacts to be randomly sampled, and that can provide interpretable feedback to a designer. Bayesian Networks (BNs) provide such a technique.

BNs are directed acyclic graphs that represent relationships of conditional probability distributions. By encoding high level design decisions as Random Variables (RVs) in the BN, it can learn how something like level size impacts the design of the level. A designer can then set a specific value, or *observe* it in probability parlance, and then sample the rest of the network to generate a level. A designer can sample it repeatedly to generate levels that are different but similar in a probabilistic sense. Finally, because each node in a BN is an RV, it can tell us how likely any given facet of a level is, providing feedback to a designer on elements that might be out of place, due to their extreme unlikeliness.

Our contribution is a probabilistic data-driven approach to level generation that allows for authorial control. We learn a probabilistic BN model of level topology (how different types of rooms are connected together) that captures the

distributional information in hand-authored example levels. This model can then be sampled to generate a new dungeon topology.

2. RELATED WORK

Level generation using machine learning techniques is mostly an unexplored area. Roberts et al. [15] used discriminative machine learning to learn player models which in turn were used as a recommender system to evaluate new levels. Dahlskog et al. [6] used Markov chains to generate Mario style platformer levels from n-grams learned from the Super Mario Bros. 1 corpus. This technique allows for the generation of new levels, but only provides authorial control in determining the types of levels generated by means of corpus choice (e.g., to generate only "underground levels" the author would have to make sure to train on only "underground levels"). Shaker and Abou-Zleikha [16] used non-negative matrix factorization to learn from the output of other level generators. They were then able to produce levels that had similar characteristics to the existing levels and also used their generator to generate levels in "unbounded" parameter space. However, the only control afforded to authors is the ability to interpolate between known existing levels or to explore the parameters space randomly.

However, there has been research into the application of probabilistic, generative machine learning techniques to problems in the field of residential floor plans [13]. Architectural design of residential floor plans is analogous to the design of ARPG levels, as both are segmented into rooms connected by doors that have specific purposes, and the design must consider both the physical and player (or occupant) space. Machine learning has also been used in other areas of procedural content generation, perhaps most impressively with the use of convolutional neural networks to learn images of chairs that then allows for interpolation between different types and orientations of chairs [8].

3. DATA

There are many possible probabilistic data-driven machine learning approaches that could be used in the context of ARPG level design. We could extend the work of Roberts et al. [15] and learn player models that could then be used to evaluate generated levels, but these learned player models are black boxes that do little to further our understanding of level design. We could train a system to classify levels, but this has the problem of only learning how to label levels. What is desired is a system that will allow a designer to specify high-level design considerations such as level size, desired difficulty, or general layout and then can generate new levels that fit these constraints while being probabilistically likely. A BN fulfills all of these criteria as a designer can observe a value for a distribution and the unobserved posterior distributions of the other nodes will update.

To train the BN we have annotated existing ARPG levels to be able to extract the relevant features needed. To do this, we have used level images from three different ARPGs from the Legend of Zelda series (Fig. 1). The Legend of Zelda is the progenitor of the genre as well as the genre's most popular series, and, furthermore, is the most prolific series of ARPGs with 17 titles over the course of 28 years. We have used the levels from *The Legend of Zelda*, *The Legend of Zelda: Link to the Past*, and *The Legend of Zelda: Link's Awakening*. We manually annotated 38 levels from the three games and held out 4 levels for our test set. The held out set contained one level from *The Legend of Zelda: Link to the Past* and *The Legend of Zelda: Link's Awakening* each, and two levels from *The Legend of Zelda*. The 34 levels that the model was trained on were composed of 1031 rooms in total, which was the final size of our training set. To annotate these levels, we first have to acquire them in a format that can be annotated. Fans of the series have posted images of the levels that show the physical structure of the levels as well as the placement of enemies, items, puzzles, and traps, allowing us to see the full structure of the levels [1, 2, 3, 4]. We then annotate the images to turn them into the graph topology that makes up the *physical space* of the level, where each room is represented as a node in the graph. Nodes (rooms) are annotated by what types of objects it contains: *Start*, *Enemies*, *Puzzles*, *Beneficial Items*, *Keys*, *Big Key*, *Key Item*, *Boss Enemy*, *End*. A room can contain any number of these elements, although in practice only contain up to 4 of these types of objects. The connections between rooms are annotated with one of the following directed link types: *Door*, *Bombable Wall*, *Locked Door*, *Soft-Locked Door*, *Big Key Locked Door*, *Key Item Locked Door*, *One Way Door*, *Can See The Other Room*. The *Can See The Other Room* feature is obviously not a standard door but encapsulates an important concept in ARPGs, mainly that the player is often made aware of where they need to go by getting glimpses of another room, but there is no passable edge between their current room and that room.

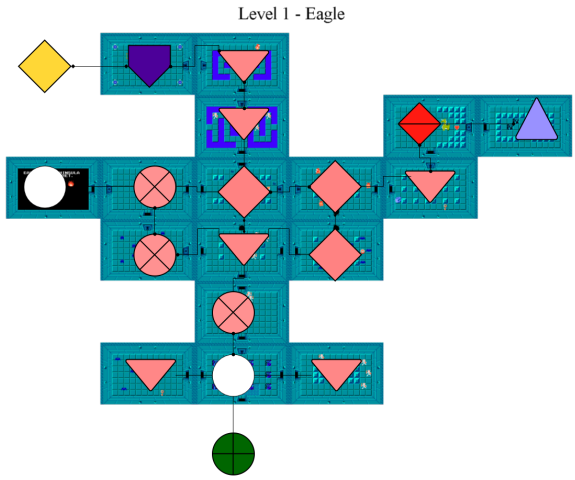

Figure 1: Annotations for the first level from *The Legend of Zelda*. Green Circle with Plus is the start point. Light Blue Upwards Triangle is the end point. Pink nodes (Diamond, Downwards Triangle, and Circle with X) contain enemies. Downward Triangle nodes contain keys. Diamond nodes contain items. Purple Pentagon nodes contain puzzles. The Red Diamond with horizontal line node contains the boss. The Yellow Diamond off the map shows that it warps to another room

After annotating the levels, we then need to extract the relevant features as the raw graph structure of the level does not provide the proper level of abstraction, as well as the

fact that raw structure only considers the physical space of the mission. To do the feature extraction we parse the level graph and then extract the *player space* of the level. To do this, we calculate an optimal path through the level from which we can then calculate the relevant features that we need to understand the physical, player, and mission spaces of the level.

3.1 Graphical Model

Once the features have been extracted, we need to construct the BN that best captures the dependencies. To do this we must first determine the underlying distributions for the features.

The categorical features, i.e. room type and door type features, are considered in a bag of words style and have a Dirichlet prior distribution. Given that a room in *Zelda* can be any combination of the aforementioned types, e.g., a room could contain a puzzle, enemies, and treasure, we need to consider these features together. Numerical types of features, such as number of rooms in the level and the length of the optimal path through the level, are treated as Gaussian distributions. If a node has categorical parent nodes, a table of posteriors is learned, one for each combination of parent category values. If a node has Numerical parents, then a regression is learned, either a Gaussian or SoftMax regression if the child is Numerical or Categorical, respectively. Depending on the combinations of parent and child distribution types there are a number of different possible posteriors that can be learned as shown in Table 1.

In total, we considered 63 different features 33 of which were global level and 30 of which were local. The global level features have considerations that deal with the entirety of a level. Some of these global features are of the *physical* space such as the number of rooms in a level or the proportions of room types within a level. Some are of the *mission* space such as whether the dungeon contains a special item or boss key. Finally, some are of the *player* space such as the length of the optimal path, and the number of times the player must backtrack to complete the level. The local features are on a per-room basis and consider features such as the room type, the neighbors of the room, whether it lies, and the types of connections to its neighbors. As a side note, due to the fact that distributions are learned for all possible combinations of parent categories, we limit the number of categorical parents to 2. This reduces the combinatorial explosion caused by considering multiple categorical parents and lessens the burden on our limited dataset.

	Child Distribution	
Parents	Categorical (C)	Numerical (N)
No Parent	$P(C)$ Dirichlet	$P(N)$ Gaussian
Categorical	$P(C\|C_1, C_2))$ Dirichlet Table	$P(C\|C_1, .., C_i))$ Gaussian Table
Numerical	$P(C\|N_1, ..., N_i))$ SoftMax	$P(N\|N_1, .., N_i))$ Linear Regression
Mixed	$P(C\|N_1, .., N_i, C_1, C_2))$ SoftMax Table	$P(N\|N_1, .., N_i, C_1, C_2))$ Linear Regression Table

Table 1: Modeled Distributions for Parent-Child Combinations

With these building blocks we constructed five graphical models. Three represent our best guess at the important features and the relationships between them, while two are only considered for baseline purposes.

- *Full* - Has 63 nodes and 97 edges. Contains all of the considered features and represents our inductive bias as to how best combine the features.
- *Naïve Bayes* - Has 63 nodes and 62 edges. Number of rooms is the variable that all distributions are conditioned on. Considered only as a baseline.
- *Random* - Has 63 nodes and 100 edges. The edges are assigned at random. Considered only as a baseline.
- *Sparse* - Has 40 nodes and 69 edges. The nodes and edges are a strict subset of the *Full* model and removes some of the nodes that are of secondary importance, e.g., only considers the number of neighbors that contain enemies or puzzles, but not the number that contain items.
- *Extreme Sparse* - Has 13 nodes and 29 edges. The nodes and edges are a strict subset of the *Sparse* model. It contains none of the nodes of secondary importance.

We included the *Naïve Bayes* model to examine whether the data did indeed have conditional dependencies. The *Random* is included to verify that our intuitions about the structure were meaningful and that a random guess would not perform similarly.

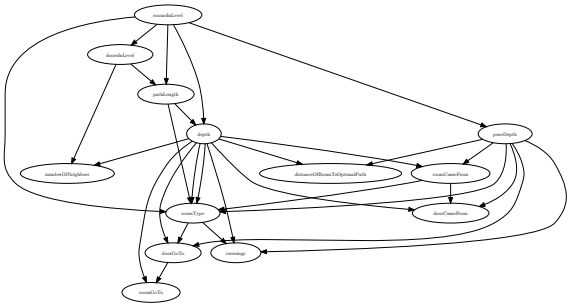

Figure 2: The Extreme Sparse Model

The smallest model, *Extreme Sparse*, can be seen in Figure 2. Due to the fact the Bayesian Information Criterion (BIC) penalizes model complexity we wanted to have a range of models covering various levels of complexity to determine what the best compromise between simplicity and covering the desired features. Table 2 reports the BIC scores for our various network topologies after training.

We used the Infer.NET probabilistic programming framework from Microsoft Research [14] which uses the Variational Message Passing (VMP) protocol to infer the underlying distributions given a graphical model and observed data. To train the network we set up the relationships between nodes, set the observed data, and then infer the posterior distributions.

4. MODEL VALIDATION

To evaluate our models, we used a modified version of the BIC introduced in [10] to obtain a single number representing the log-likelihood of the training data given our model with a regularization term to penalize model complexity as seen in the formula.

$$\log p(D|S^h) \approx \log p(D|\hat{\theta}_s, S^h) - \frac{d}{2} \log N$$

A perfectly likely model would have a log-likelihood of 0, i.e. a probability of 1. If there were multiple such models, the simplest one, i.e. the one with the smallest regularization penalty, would be chosen. More commonly, there is a trade off between a model that better explains the data, i.e. is more likely, but that is more complex, i.e. has a higher regularization term. The less likely a model is, the lower the log-likelihood will be, so a BIC closer to 0 is better than one that is more negative.

Learning Method	BIC
Full	$-1.656e^7$
Naïve Bayes	$-3.224e^8$
Random	$-1.781e^{11}$
Sparse	$-1.618e^7$
Extreme Sparse	-1.581^7

Table 2: BIC scores for the different network topologies

As expected the *Random* model performs the worst, performing orders of magnitude worse than any of the structured models. The *Naïve Bayes* model is an order of magnitude worse than any of other structured models, which would indicate that the features do indeed have some conditional dependence. The *Full* model performs at a rate slightly worse than the *Sparse* or *Extreme Sparse* models, but the differences between the three are negligible.

To test the predictive power of our learned model against the training data and the Sum of the Square Error (SSE) and the Average Error per level can be seen in Table 3. We used the training data, but removed *Number of Rooms in Level* which was then inferred. As you can see, the Naïve approach does demonstrably worse than the graphical approaches. As might be expected from the similarity of the BIC values, the *Full*, *Sparse*, and *Extreme Sparse* preform similarly. For the purposes of prediction, the *Extreme Sparse* preforms the best.

Learning Method	SSE	Avg. Error
Full	284.8	2.89
Naïve Bayes	2037.8	7.74
Sparse	258.3	2.76
Extreme Sparse	255.4	2.74

Table 3: SSE of Inferred Room Count from Training Data

Looking at the held out test set in Table 4 we see that our networks do not perform as well as they did on the training data. Performing paired t-tests between the different models, we see that except for *Sparse* vs *Full* each model performs differently at a statistically significant $\mathbf{(p < 0.01)}$ level. Our randomly held out test data were atypical and as such speak to the difficulty of learning on such a small amount of varied data. The level chosen from *Link to the Past* happened to be the largest level considered and as such was very much an outlier compared to the rest of our data. If it were ignored the average errors decrease to be much closer to the training results.

Learning Method	SSE	Avg. Error	SSE*	Avg. Error*
Full	163.9	6.40	34.9	3.41
Naïve Bayes	581.3	12.06	224.5	8.65
Sparse	160.1	6.3	33.3	3.33
Extreme Sparse	157.0	6.24	35.0	3.42

Table 4: SSE of Inferred Room Count from Test Data
* the outlier has been removed.

5. CONCLUSION AND FUTURE WORK

In this paper we have presented a method for automatic learning of design decisions from human authored levels. We then showed that the trained networks could be used as a form of classifier based on the learned relationships. While we as designers had to insert our biases about the variables that would be interesting/valuable, we did not need to decipher and then encode any design decisions ourselves. That being said, we did hand produce the BN topologies. In future work, we would like to apply network learning techniques so that even the relationship between variables was machine learned.

Not shown in this paper is how the trained networks could be used for the generation of new artifacts. We have already started generating levels using the networks learned from our training set. Currently, this works by a designer specifying high level constraints, such as number of rooms, and then the system samples until it has a complete, valid level. In future work, we would like to generate these levels using a Reversible-Jump Markov Chain Monte Carlo system. By sampling over the space of generated levels using a Monte Carlo process we can generate levels that match our desired distribution, i.e. the levels that we learned from in the first place.

Our existing work only considered level topology, but the same principles can be applied at finer granularity. Specifically, we could learn design decicions for tile placement. On that note, there are many games for which level topology is an uninformative space and would need to be modeled at tile-placement granularity. It is our hope to extend our work to this level.

Another possible path for this would be a mixed initiative tool, in line with the work of Butler, et. al [5] or Smith, et. al [17]. Just as decorative elements are important to later Zelda games, most later Zelda games have "showpiece" rooms that are especially elaborate and have very specific theming. We could imagine an authoring tool that would allow for a designer to specify high level desires about the dungeon that could incorporate generated "filler" rooms that work well with the human authored "showpiece" rooms. A mixed initiative tool based on probabilistic principals could help guide a designer by highlighting unlikely sections of the level that might be problematic.

6. REFERENCES

[1] The legend of zelda world & dungeon maps, Nov. 2014. http://zs.ffshrine.org/legend-of-zelda/maps.php.

[2] The legend of zelda oracle of ages, Nov. 2014. http://www.angelfire.com/games4/ages/.

[3] Link's awakening maps, Nov. 2014. http://www.zeldaelements.net/games/c/links_awakening/maps.

[4] I. Albert. Legend of Zelda Maps, Nov. 2014. http://ian-albert.com/games/legend_of_zelda_maps/.

[5] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 377–386, New York, NY, USA, 2013. ACM.

[6] S. Dahlskog, J. Togelius, and M. J. Nelson. Linear levels through n-grams. In *Proceedings of the 18th International Academic MindTrek Conference*, 2014.

[7] J. Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 1:1–1:8, New York, NY, USA, 2010. ACM.

[8] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CoRR*, abs/1411.5928, 2014.

[9] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. Toward supporting stories with procedurally generated game worlds. In S.-B. Cho, S. M. Lucas, and P. Hingston, editors, *CIG*, pages 297–304. IEEE, 2011.

[10] D. Heckerman. Learning in graphical models. chapter A Tutorial on Learning with Bayesian Networks, pages 301–354. MIT Press, Cambridge, MA, USA, 1999.

[11] I. D. Horswill and L. Foged. Fast procedural level population with playability constraints. In M. Riedl and G. Sukthankar, editors, *AIIDE*. The AAAI Press, 2012.

[12] R. v. d. Linden, R. Lopes, and R. Bidarra. Designing procedurally generated levels. In *Proceedings of IDPv2 2013 - Workshop on Artificial Intelligence in the Game Design Process, co-located with the Ninth AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment*, pages 41–47, AAAI Press, Palo Alto, CA, oct 2013. AAAI, AAAI Press. ISBN 978-1-57735-635-6.

[13] P. Merrell, E. Schkufza, and V. Koltun. Computer-generated residential building layouts. *ACM Trans. Graph.*, 29(6):181:1–181:12, Dec. 2010.

[14] T. Minka, J. Winn, J. Guiver, and D. Knowles. Infer.NET 2.5, 2012. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[15] J. Roberts and K. Chen. Learning-based procedural content generation. *CoRR*, abs/1308.6415, 2013.

[16] N. Shaker and M. Abou-Zleikha. Alone we can do so little, together we can do so much: A combinatorial approach for generating game content. In *AIIDE'14*, pages –1–1, 2014.

[17] G. Smith, S. Member, J. Whitehead, S. Member, and M. Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, page 2011.

[18] N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. In *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*, EvoApplicatons'10, pages 131–140, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] V. Valtchanov and J. A. Brown. Evolving dungeon crawler levels with relative placement. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 27–35, New York, NY, USA, 2012. ACM.