# Sampling Hyrule: Sampling Probabilistic Machine Learning for Level Generation

**Adam Summerville and Michael Mateas**
Expressive Intelligence Studio
University of California, Santa Cruz
asummerv@ucsc.edu, michaelm@soe.ucsc.edu

**Abstract**

Procedural Content Generation (PCG) using machine learning is a fast growing area of research. Action Role Playing Game (ARPG) levels represent an interesting challenge for PCG due to their multi-tiered structure and nonlinearity. Previous work has used Bayes Nets (BN) to learn properties of the topological structure of levels from *The Legend of Zelda*. In this paper we describe a method for sampling these learned distributions to generate valid, playable level topologies. We carry this deeper and learn a sampleable representation of the individual rooms using Principal Component Analysis . We combine the two techniques and present a multi-scale machine learned technique for procedurally generating ARPG levels from a corpus of levels from *The Legend of Zelda*.

## Introduction

Procedural Content Generation has seen wide spread adoption in the games research community as well as the video game industry. In large part, the methods for procedural generation of video game levels have focused on designers trying to codify and proceduralize their design decisions to create a generator. However, this is a difficult process and a designer might unknowingly encode hidden biases or might not have a full grasp of their intuitive design process. We feel that a better source for a designer's design knowledge are the artifacts that they themselves generate as they inherently encode the designer's decisions and knowledge. Towards this end, we use previous work by Summerville et al. (Summerville et al. 2015) as a machine learned representation of design knowledge for Action Role Playing Game (ARPG) levels from *The Legend of Zelda* series and present a system for generating levels that have the same statistical properties as the human-authored levels.

Previous applications of machine learning techniques towards the goal of procedural content generation have mostly been focused on learning to generate levels for platformer games, specifically those in the *Super Mario Bros.* series. These have ranged from applications of Markov chain generation (Dahlskog, Togelius, and Nelson 2014a; Snodgrass and Ontanon 2013), to learning graph grammars (Londoño and Missura 2015) via automatic grammar induction techniques. Platformer level generation has been the most popular use of machine learning, but ARPG levels present an interesting challenge for procedural content generation. Platformer levels tend to be highly linear, typically progressing from left-to-right. All known uses of machine learning for platformer level generation have relied on this property (although levels in other platformer games are not guaranteed to have this property). On the other end of the spectrum, ARPG levels tend to be highly nonlinear. They also tend to combine high level mission parameters (e.g. find a key to unlock a door, find an item to solve a puzzle, etc.) with a large amount of player backtracking through a densely connected space. The only known application of machine learning techniques to *The Legend of Zelda* is the work of Summerville et al.

The learned representation that we use operates at two discrete scales. At the highest level is a Bayes Net (BN) that represents the graph topology of the room-to-room connections in ARPG levels. In and of itself, it contains features that operate a number of different scales. These range from low level room-to-room properties to high level properties of the level as a whole, such as what is the length of the optimal path through the level. At the 50,000 foot level ARPG levels can be thought of as directed graphs with rooms as nodes and doors as edges. The BN was trained on this representation and is then used to generate a graph representation of the level. While this representation encodes the aspects of ARPG levels that make them an interesting challenge, it leaves out a crucial aspect of the levels, the rooms themselves. Towards this end, we present a machine learned representation of the rooms using Principal Component Analysis (PCA) for feature compression. We then interpolate between human authored levels in this reduced feature space to generate rooms that have the desired properties (a room that contains enemies, a key, and a puzzle). Our contributions are a novel use of sampling BNs to generate playable ARPG level graphs that are then made concrete by interpolating within a PCA compressed feature space.

## Related Work

Machine learned representations of video game levels have taken a number of approaches. Despite the breadth of techniques used, the vast majority have been stochastic in nature. Markov chains have been the preferred method for probabilistic platformer level generation. Dahlskog et al. (Dahlskog, Togelius, and Nelson 2014b) used Markov chains to generate Mario style platformer levels from n-grams learned from the Super Mario Bros. 1 corpus. Dahlskog chose to represent levels as a series of vertical slices and learned transmission probabilities from one vertical slice to the next looking at up to trigram level historicity. Snodgrass and Ontan (Snodgrass and Ontanon 2013) similarly used Markov chains as their method for level generation. However, instead of looking at vertical slices they used a 2-dimensional Markov chain that operated at the level of individual tiles. Both of these techniques are capable of generating new levels, but the only level of authorial control is in the selection of the training corpus.

Guzdial and Riedl (Guzdial and Riedl 2015) used a probabilistic model with latent features to capture similar tile blocks in Mario levels. They then learned relationships between tile groups for placement purposes. This has a benefit over Markov chains in that it does not require an extremely large corpus to provide information at a larger scale. For instance, Dahlskog was only able train a Markov chain up to the trigram level, providing 3 tiles of information, whereas Guzdial's work was able to learn relationships that spanned upwards of 18 tiles in distance. Londoo and Missura (Londoño and Missura 2015) similarly used a system that learned relationships between tiles, but instead used graph grammar induction as their method of learning. Unlike the other work, they eschewed a representation focused on physical relationships (tile X is next to tile Y) instead opting for semantic relationships (tile X is reachable from tile Y or tile X is connected to tile Y via a platform). While not yet used for actual level generation the use of semantic relationships could perhaps lead to a wider expressive range of generation.

Deterministic machine learning techniques are uncommonly used in the realm of level generation. In part this is due to the fact that a probabilistic representation is appealing for level generation. Nonetheless, Shaker and Abou-Zleikha (Shaker and Abou-Zleikha 2014) used non-negative matrix factorization to learn a compressed feature expression of Mario platformer level generators. Non-negative matrix factorization is a technique that factorizes a source matrix into two matrices that can broadly be thought of as a feature representation and a set of weights. It has the benefit that all of the matrices involved are guaranteed to be positive. The nonnegative portion makes sense in the domain of platformer tilemaps, as a tile either exists (positive) or not (zero), but the concept of a negative tile has no meaning.

## Machine Learning Techniques

Levels in the 2D members of *Legend of Zelda* series are a series of connected rooms. Due to this natural segmenting of rooms, we used two separate machine learning paradigms. Each technique has a different set of affordances that makes it useful for its targeted problem area.

### Bayes Nets

At the level of room topology, Bayes Nets (BNs) were chosen as the machine learning technique. BNs learn relationships between feature distributions and upon observing data can be used to infer the rest. In the previous work by Summerville et al. a number of different models were tested with the smallest model, the *Sparse* model, having not only the best tradeoff between predictive power and complexity, but flat out having the best predictive power. This is the model that we chose to use for generation purposes and can be seen in figure 1. The trained BN captured a variety of high level features, such as number of rooms in the level and length of optimal path through the leve, along with low level features such as room-to-room connections and room types. At generation time a designer can set whatever they want (or rather *observe* in the probability parlance) and the system will work with that and infer the rest. e.g. a designer could observe the size of the level, the general make up of rooms, etc.. The system could even be used in a mixed initiative manner with a designer laying out a set of rooms and allowing the system to fill in the rest. The learned network was composed of the following features:

- *Number of Rooms in Level*
- *Number of Doors in Level*
- *Number of Path Crossings in Level*
- *Length of Optimal Path*
- *Distance of Room to Optimal Path*
- *Path Depth of Room in Level*
- *Distance From Entrance*
- *Room Type*
- *Previous Room Type*
- *Door Type from Previous Room*
- *Next Room Type*
- *Door Type to Next Room*

and was trained on 38 levels from **The Legend of Zelda**, **The Legend of Zelda: A Link to the Past**, and **Legend of Zelda: Link's Awakening** that comprise a total of 1031 rooms.

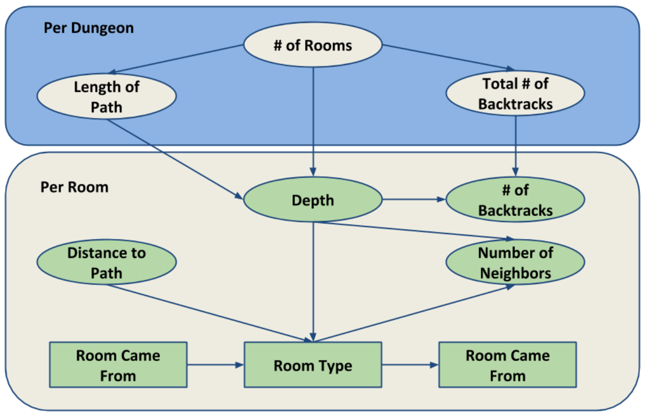


Figure 1

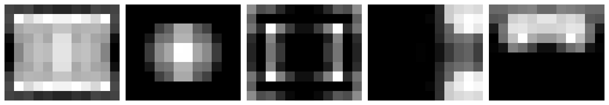


Figure 2: The first 5 Principle Componenets of the solid blocks. These account for 70% of all variation

### Principal Component Analysis

Principal Component Analysis (PCA) is a technique that finds a set of orthogonal features that best capture the covariance between the original feature set. PCA has been used for feature recognition and dimensionality reduction for faces in a technique known as *Eigenfaces* (Turk and Pentland 1991). Using *Eigenfaces* and the work of Shaker and Abou-Zleikha as inspiration we treated rooms as binary bitmaps. While an actual tilemap is not binary, it is better to treat it in a one-hot manner with multiple channels of information. The tile types that we considered were:

- *Solid*
- *Enemy*
- *Item*
- *Key*
- *Trap*
- *Puzzle*
- *Water/Bottomless Pit*
- *Key Item*
- *Teleporter*

Different graphical representations or sub-representations (i.e. is an enemy a bat or a skeleton) are ignored and only the highest level of purpose is considered. Each of the 9 tile types is treated as its own binary images and were treated separately when performing the PCA.

We used 488 rooms from the three aforementioned games for the PCA. However, to increase the size of the sample space we also used all mirrorings (up-down, left-right, both) to quadruple the size of our dataset. Unfortunately, rooms from the three games differ in size so we had to find a way to rescale the rooms to fit the maximum horizontal (12) and vertical sizes (10). To do this, a preprocessing step was performed where we performed a graph cutting procedure on the dimensions that needed to be expanded. To do this we found a path of minimal transitions from filled to unfilled and vice-versa and expanded along this line.

For PCA on images each pixel, or in this case tile, is treated as its own feature. Given the $12 \times 10$ room size we had a total of 120 features. Upon performing PCA the new feature set is ranked upon the amount of variance from the original feature set that it accounted for. There is no general rule for what the desired level of feature compression is, but we decided that the minimal feature set that could perfectly recreate our original rooms would suffice. In the end this meant that we accounted for approximately 95% of the original information with a compressed feature set of 20 features. The first five principal components of the solid tile type can be seen in figure 2.

Like Non-Negative Matrix Factorization (NNMF), PCA splits the original feature matrix into 2 sub-matrices that can be thought of as the compressed feature set and the weights to apply to those features. PCA tends to learn global features, unlike NNMF which tends to learn local features. During the research process we performed a comparison between PCA and NNMF and found PCA performed much better than NNMF for this domain. Where PCA was able to reduce the feature set by a factor of 6, it took the entire "reduced" NNMF feature set to accurately recreate the original rooms.

With the reduced feature set we can then interpolate between the weight vectors of different rooms to find rooms that are in between the originals. Our original intention was to simply find 2 original rooms that had the features that we wanted (e.g. a room that had items and enemies) and interpolate between them. However, we found that blindly taking rooms, even of the same type, was no guarantee of success when interpolating. A large amount of rooms looked like they had been created from random noise, lacking any of the structure that we would have hoped to have been captured during the PCA process. To combat this, we performed a clustering step where rooms were clustered based on their weight vectors using k-Means clustering. This ensured that the rooms being chosen to interpolate between were at similar places in the 20-dimensional space, which eliminated the problem of interpolating through ill defined portions of the space.

## Level Generation

With the learned representations of the level, we need to actually generate levels. We first construct the room-to-room topology of the level, and then go through on a per-room basis and generate the rooms using the PCA representation.

### Level Topology

When generating the level topology a designer can first observe any design features that they wish. For the purposes of our generation we only observed the size of the levels. We then sample the learned distributions for the high level, per-dungeon parameters and observe them to fix them throughout generation. As of now, only the number of rooms is treated as a hard constraint, i.e. if the designer specifies that there are 31 rooms in the level then it is guaranteed that there will be 31 rooms in the level. The rest are treated as soft constraints that only inform the dependent distributions.

Once the global level parameters have been inferred, the dungeon is built up from a seed room. The initial entry point to every dungeon is a special *Start* room. *Start* rooms only ever have 1 neighbor, and it is this first neighbor room that is inferred. The room type, the incoming door type, and the number of neighbors are all inferred, given the prior room type (initially the *Start* room), the traversed door type (initially a regular door), its depth in the dungeon (initially one), and all of the inferred or specified global parameters. During this process there is a simultaneous grid embedding process that happens in parallel. The *Start* room is always placed at $(0, -1)$ and its child room is always placed at $(0, 0)$. Each new child room is placed at an open neighboring place on the grid, and added to a list of rooms to be inferred. Once all of the specified number of rooms have been placed, a second cleaning process guarantees that the level is playable.

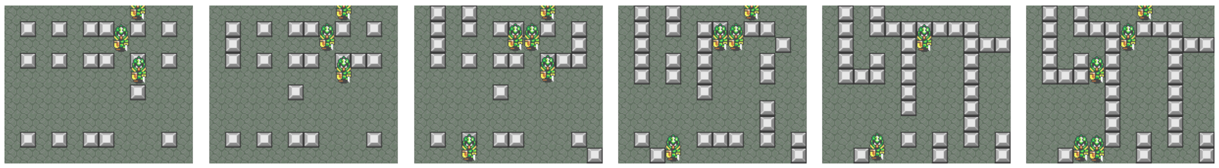


Figure 3: Interpolation between two human authored rooms. The rooms on the ends are human authored while the rooms in between represents steps between them of 20%.

For a level to be playable, all of the rooms must be accessible, the number of key locked doors must equal the number of placed keys, all keys must be used to complete the level, all keys must be in front of the doors they lock, and there must be a special item room, a boss room, and an end room. To resolve any of these constraints, we utilize the same machinery that tested the validity of the algorithm, the log-likelihood of a specific event occurring. For example, if there was no end room placed, we simply iterate over all rooms, find the one that has the highest likelihood of being an end room, and change it to an end room. We do this for the boss and special item rooms, in addition to the key locked doors. Once these have been taken care of, we then find the optimal path through the dungeon in order to guarantee that the optimal path through the dungeon uses all of the placed keys. If the level is not completable, then we know that the player is unable to reach a required key. This can be resolved in one of two ways, **1)** Move a key that has not been reached to a room that has been reached, or **2)** Move a locked door that has been reached to a door that has not been reached. The choice of how to resolve the violation is chosen at random. On the other hand, if the level is completable, but not all keys are used, then a key-locked door is placed unnecessarily. To resolve this, an unseen key locked door is moved to a seen unlocked door. No matter what constraint violation is being resolved, it is handled with the same machinery seen above, e.g. if a key needs to be moved, then the existing key room that is least likely to be a key room is removed, and the room it is moved to is the one most likely to be a key room.

## Room Generation

With the level topology constructed we need to generate the rooms at the tile level to have a playable level. To generate a room of a given type, we find a cluster that contains at least two instances of that type, choosing two at random, and interpolating between the two randomly. Despite the clustering cutting down on unplayable rooms, it is not guaranteed that a generated room will be playable. Moreover, it is possible that a generated level might not have the desired properties, e.g. an interpolated room contains no enemies despite the two source rooms containing enemies. Another concern is that the generated room might have obstructions to the required doors. If there are any violations, the generated room is thrown out and a new one is sampled. While not ideal, this process is very fast and even with oversampling takes under a second. An example of the interpolation process can be seen in figure 3

## Results

A representative sample of generated levels can be seen in figure 4. While all generated levels are guaranteed to be completable, it is an open question as to how good they are. Due to the dual nature of our algorithm as both a classification tool and a generation tool, it is able to evaluate the artifacts that it produces. To perform this evaluation use the Bayesian Information Criteria (BIC) to evaluate how likely the generated levels are.

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

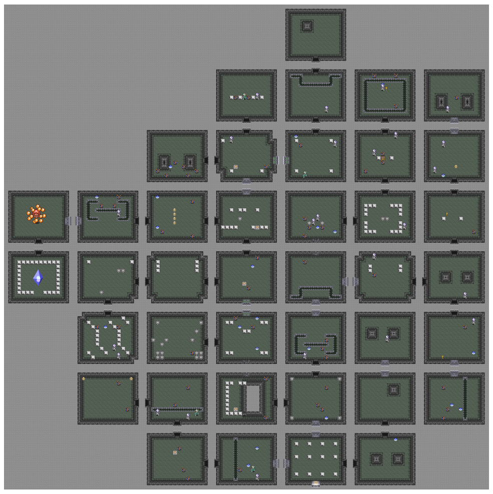
where $D$ is the data of the model, $S^h$ is the chosen model, $\hat{\theta}_s$ are the parameters of the model, $d$ is the number of free parameters in the model, and $N$ is the number of datapoints. The BIC has two terms, the first is based on the likelihood of the data given the learned model, the second is a penalty term that penalizes the model based on how many terms it has (i.e. how complex it is). The BIC works well to evaluate the levels as it can tell the likelihood of a generated level; However, given that the regularization term imposes a penalty for each data point seen we can only compares levels of the same size to each other. In reality this is not a major concern, as the likelihood of a level does not make much sense in the abstract and is only important as it relates to comparing the likelihood of two levels. As such, we compare the likelihood of our generated levels to human authored levels, considering 3 different sized levels. The sized chosen are:

- *Small* - Levels contained 12 rooms, the size of the smallest level in our dataset.
- *Medium* - Levels contained 35 rooms. The mean size of levels in the dataset was 33.9. The closest level comprises 35 rooms.
- *Large* - Levels contained 47 rooms. The standard deviation for room size was 14.2, so this represents one standard deviation from the mean size.

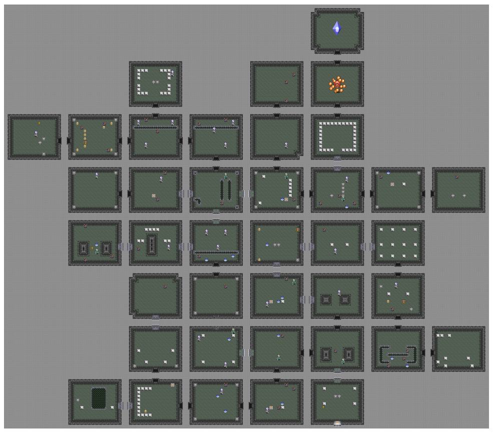
For the purpose of testing we produced 20 levels of each size class, and the results can be seen in table 1.

| Level Size | Authored BIC | Maximum Generated BIC | Mean Generated BIC |
|---|---|---|---|
| *Small* | -14,568 | -15,171 | -15,444 |
| *Medium* | -25,006 | -28,904 | -32,483 |
| *Large* | -38,722 | -39,361 | -43,813 |

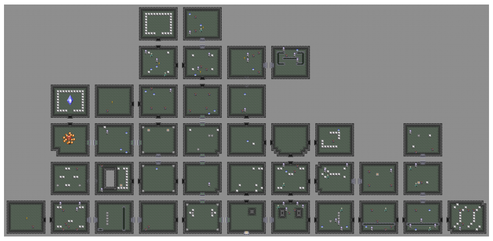
Table 1: Comparison of log-likelihoods between human authored levels sand generated levels.



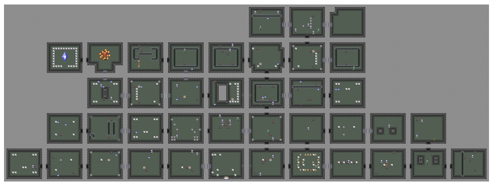
(a)



(b)



(c)



(d)

Figure 4: Representative sample of 4 generated levels

As can be seen, the human authored levels are seen to be more likely, but this is not surprising. Currently, a number of different factors are working against the generated levels. Most importantly, the human authored levels that were tested against were part of the training set, making them quite likely according to the model. Furthermore, since only room size is treated as a hard constraint, a generated level might not have the desired properties. Finally, due to the nature of how level violations are resolved the generated levels are treated in a way that maximizes likelihood in a local greedy manner. If the violations were resolved maximizing likelihood for the entire level, the end results would almost certainly have a higher likelihood.

## Conclusions and Future Work

Procedural content generation via machine learned techniques are an open area of research with many different possible approaches. In this paper we presented a hybrid approach that uses two machine learning techniques to generate levels in the style of *The Legend of Zelda*. Bayes Nets present an appealing representation for learning high level design knowledge for procedural content generation due to the ability for a designer to *observe* parameters for generation and allow the system to infer the rest. Principal Component Analysis, used in the style of Eigenfaces, can be used to learn orthogonal, correlated features in images. Applying this to tilemaps we can then interpolate between existing rooms to generate new rooms that are similar to human authored levels.

Evaluation techniques for video game levels, specifically procedurally generated levels, is an open area of research. In this paper we present a unique evaluation approach. Instead of creating a separate metric (or suite of metrics) we compare how likely our generated levels are compared to human authored levels. Given the goal of learning design knowledge for the purpose of generation, we can directly use that learned knowledge as an evaluation scheme. While the generate levels perform more poorly than the existing human authored levels, this is to be expected as the human authored levels were part of the training corpus. In future work we would like to compare other levels from *The Legend of Zelda* series and fan-authored levels to see how they compare to our generated levels.

## References

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

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

Guzdial, M., and Riedl, M. O. 2015. Toward game level generation from gameplay videos. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*.

Londoño, S., and Missura, O. 2015. Graph grammars for super mario bros levels. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*.

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

Snodgrass, S., and Ontanon, S. 2013. Generating maps using markov chains.

Summerville, A. J.; Behrooz, M.; Mateas, M.; and Jhala, A. 2015. The learning of zelda: Learning to design levels for action role playing games. In *Proceedings of the 10th International Conference on the Foundations of Digital Games*.

Turk, M., and Pentland, A. 1991. Eigenfaces for recognition. *J. Cognitive Neuroscience* 3(1):71–86.