A Hierarchical MdMC Approach to 2D Video Game Map Generation

Sam Snodgrass and **Santiago Ontañón**
Drexel University, Department of Computer Science
Philadelphia, PA, USA
sps74@drexel.edu, santi@cs.drexel.edu

Abstract

In this paper we describe a hierarchical method for procedurally generating 2D game maps using multi-dimensional Markov chains (MdMCs). Our method takes a collection of 2D game maps, breaks them into small chunks and performs clustering to find a set of chunks that correspond to high-level structures (high-level tiles) in the training maps. This set of high-level tiles is then used to re-represent the training maps, and to fit two sets of MdMC models: a high-level model captures the distribution of high-level tiles in the map, and a set of low-level models capture the internal structure of each high-level tile. These two sets of models can then be used to hierarchically generate new maps. We test our approach using two classic games, *Super Mario Bros.* and *Loderunner*, and compare the results against other existing map generators.

Introduction

Generating video game maps algorithmically (called procedural content generation, or PCG) allows players to experience new and unique content. Unlike most PCG methods, which include carefully encoded design and domain knowledge (Togelius et al. 2011), we are interested in using machine learning methods to automatically extract latent design knowledge from expert-crafted maps. By creating content generators that are directly shaped by training examples, we explore an important new way of controlling PCG systems.

In our previous work (Snodgrass and Ontañón 2014a), we introduced an approach based on training a statistical model from a set of given human-authored maps that could then be used to sample new maps with the same statistical properties. Specifically, we showed that multi-dimensional Markov chains (MdMCs) could be used to sample maps for two-dimensional platformers such as *Super Mario Bros.* One drawback of such approach is that, unless an unreasonable amount of training data is available, MdMCs can only capture small-scale structure in the maps. To address this problem, we proposed a hierarchical approach using used hand-authored high-level blocks to represent the high-level structure in the training maps (Snodgrass and Ontanon 2014b). This provided great improvement over the non-hierarchical approach, but required too much domain knowledge. This paper focuses on the automatization of that manual hierarchical approach, including an extension to a more complex domain, as well as a significant extension to the evaluation methodology. We introduce a method that automatically finds the high-level structures of a map through clustering. Once we find the high-level structures, we automatically annotate the input maps with tiles representing the found structures. We then fit a high-level and a low-level model to the annotated maps and input maps, respectively. Afterwards, we sample a high-level map using the learned high-level model, and a low-level map using both the learned low-level model and the newly sampled high-level map.

We validate our approach using two different domains: *Super Mario Bros.* and *Loderunner*. Additionally, we compare the *Super Mario Bros.* maps sampled by our method against those created by the top performing generators from the 2011 *Mario AI Championship* (Shaker et al. 2011).

Background

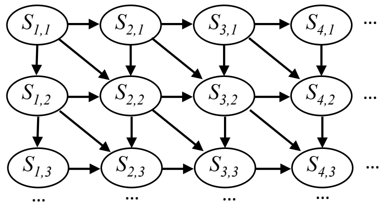
Procedural Map Generation

Procedural content generation (PCG) methods are used to create content algorithmically instead of manually (Togelius et al. 2011). For a more complete introduction to PCG the reader is referred to Hendrikx et. al. (2013).

Many PCG systems work by pairing a search technique with an evaluation function that judges the output (Togelius et al. 2011; Smith and Mateas 2011). In our approach, we are interested in generators that learn statistical properties from a set of given maps, and use them to sample new maps. Such generative methods have already been used for text and music generation (Shannon 2001; Conklin 2003); we are interested in seeing how well they translate to map generation. Shaker et. al. (2011) describe a level generation competition in which several systems employed models learned from data. However, these systems learned models of the players, not the map characteristics.

Tile grids, used in our work, are common in PCG systems as they correlate strongly to the map representations in many games. Some commercial games, for example *Civilization IV*[1] and *Spelunky*[2], include map generators that work by assembling tiles on a grid. There has also been some academic research exploring learning to generate tile-based

[1]http://www.2kgames.com/civ4/
[2]http://www.spelunkyworld.com/



$P(S_{t,r} \mid S_{t-1,r}, S_{t,r-1}, S_{t-1,r-1})$

Figure 1: A two-dimensional Markov chain of order three (three incoming dependence arrows for each node).

maps from examples (Dahlskog, Togelius, and Nelson 2014; Snodgrass and Ontanon 2014b).

Markov Chains

Markov chains (Markov 1971) are a method for modeling probabilistic transitions between states. A Markov chain is defined as a set of states $S = \{s_1, s_2, ..., s_n\}$ and the conditional probability distribution (CPD) $P(S_t|S_{t-1})$, representing the probability of transitioning to a state $S_t \in S$ given that the previous state was $S_{t-1} \in S$.

Higher-order Markov chains account for d previous states, where d is a finite natural number (Ching et al. 2013). The CPD defining a Markov chain of order d can be written as: $P(S_t|S_{t-1}, ..., S_{t-d})$. That is, P is the conditional probability of transitioning to a state S_t, given the past d states of the Markov chain.

Although Markov random fields (MRFs) have been used to reason about and synthesize textures (Levina and Bickel 2006), sampling from such models is computationally complex. Multi-dimensional Markov chains (MdMCs), such as the two-dimensional Markov chains in our work (an example of which can be seen in Figure 1), are an extension of higher-order Markov chains that have been shown to produce qualitatively similar texture outputs, while reducing the sampling complexity drastically (Levina and Bickel 2006).

Methods

In our previous work (Snodgrass and Ontanon 2014b) we showed that a two-layered hierarchical approach to map generation can greatly improve the quality of the sampled maps over sampling using a single low-level Markov chain, by capturing dependencies beyond immediate neighbors. However, that approach requires the manual identification of a set of high-level tiles capturing prototypical structures. In this paper, we present an approach which learns such high-level tiles from examples through clustering. We also show that our approach performs as well as the manual approach.

Specifically, the approach presented in this paper combines MdMCs and k-medoids clustering (Kaufman and Rousseeuw 1987) for hierarchical game map generation. Given a set of two-dimensional maps for a given video game, our method works as follows: first, we break each of the maps into chunks of a certain size. Then, we perform clustering on all of the resulting chunks, taking the centroids of the clusters to be the high-level tiles. Next, we use those high-level tiles to re-represent the input maps, and train a MdMC that can sample such high-level maps. We then train a low-level MdMC for each high-level tile. All these steps are described in turn below.

Map Representation

A map is represented by an $h \times w$ two-dimensional array, M, where h is the height of the map, and w is the width. Each cell of M is mapped to an element of S, the set of tile types which correspond to the states of the MdMC. Figure 2 shows a section of a *Super Mario Bros.* map (1) and how that section is represented as an array of tiles (2). Notice that we added sentinel tiles to the map to signify the boundaries.

Identifying High-Level Map Structures

In order to extract common high-level structures from the input maps, first we split each input map into chunks of a given size (e.g., 6×6), and then we cluster them using k-medoids. k-medoids clustering is an unsupervised learning algorithm that can group data into k clusters, using some distance metric (Kaufman and Rousseeuw 1987). The best value of k is dependent on the domain. We experimented with $k \in \{6, ..., 16\}$. We use the centroids of the resulting clusters as our high-level tiles. An interesting property of k-medoids is that it does not require *averaging* sets of instances (as k-means does), and is thus, applicable to domains, like ours, where instances are not real-valued vectors.

Once the high-level tiles have been found, we annotate each input map with high-level tile types. Each chunk in an input map is converted into a tile representing the most similar high-level tile type using the same distance metric used during clustering. Therefore, a high-level map is represented by a two-dimensional array H of size $(h/T) \times (w/T)$, where T is the size of the high-level tiles. Figure 2 shows a low-level map (2) and its high-level representation (3).

We experimented with the following distance metrics:

- **Direct**: Compares the two chunks tile by tile, and counts the number of tiles that are different.
- **Shape**: Slides one chunk over the other, and applies the **Direct** metric on the overlapping area of the two chunks, looking for the position in which the **Direct** distance is minimized after normalization by the overlapping area.
- **WeightedDirect**: Similar to **Direct**, but using manually defined costs to determine the penalty of matching tiles. For example, matching a "?-block" to a "breakable block" has a lower cost than matching it to an "empty space."
- **Markov**: Learns a tile distribution for each chunk using a standard Markov chain. Distance is then computed as the Manhattan distance between the resulting distributions.
- **Histogram**: Performs a simple count for each low-level tile type in both chunks. Distance is computed as the Manhattan distance between the resulting vectors.

Learning the Model

Given a set of input maps, and a set of k high-level tiles (extracted as described above), our method trains $k+1$ MdMCs. A high-level chain captures the distribution of high-level

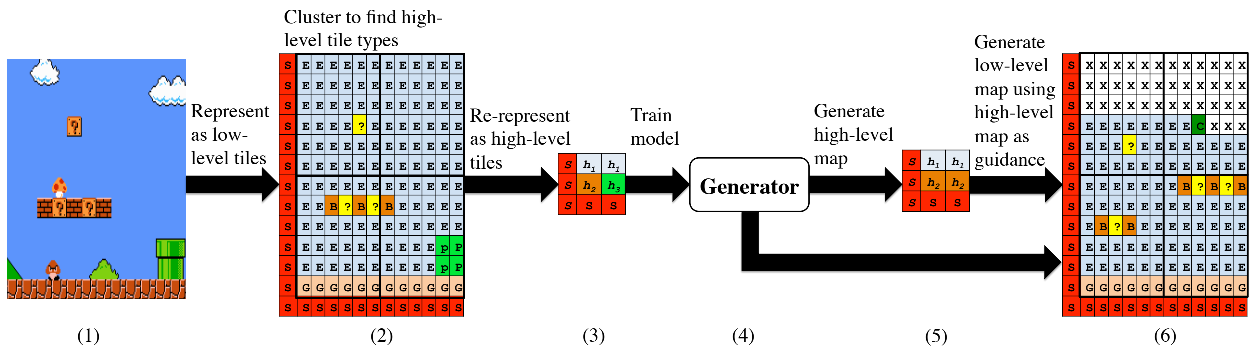

Represent as low-level tiles
Cluster to find high-level tile types
Re-represent as high-level tiles
Train model
Generator
Generate high-level map
Generate low-level map using high-level map as guidance
(1)
(2)
(3)
(4)
(5)
(6)

Figure 2: A visualization of the pipeline our approach follows. First, we represent the training maps with a set of low-level tiles (1 and 2). Next, we split the maps into $T \times T$ sized sections and perform clustering to identify high-level tiles, with which we re-represent the training maps (2 and 3). We then train a collection of MdMC models (4). A high-level map is sampled (5), and finally, we sample a corresponding low-level map (6).

tiles, and a set of k chains capture the distribution of low-level tiles within each high-level tile. The high-level model is able to capture long range dependencies by compressing many low-level tiles into a few high-level structures, while the low-level models only capture immediate dependencies.

To train a Markov chain, we need two things: 1) the dependencies between the state variables and 2) training data. Figure 3 shows the different dependency structures we use in our experiments. D_0 learns the frequency of each tile, irrespective of the neighbors; D_1 takes into account one tile to the left; D_2 takes into account one tile to the left, and one tile below; and D_3 learns the distribution of tiles given the tile to the left, the one below, and the one to the left and below.

The high-level Markov chain is trained with the input maps re-represented using high-level tiles, and the low-level Markov chain corresponding to a given high-level tile is trained with each of the chunks in the input maps that corresponds to that high-level tile. Thus, each chain receives a set of training maps, represented with tiles from a vocabulary $S = \{s_1, ..., s_n\}$, and a given dependency structure D. Training happens in two steps: *Absolute Counts* and *Probability Estimation*. First, given the dependency structure, the number of times each tile follows each previous tile configuration is counted. Next, the probabilities are set to the frequencies of the observations in the training data. For more details on this process see (Snodgrass and Ontanon 2014b).

Map Generation

Our method first samples a high-level map, then samples a low-level map. In both cases our method sample one tile at a time, starting in the bottom left corner, and completing an entire row of tiles, before moving to the next row. Low-level maps are sampled using a high-level map as guidance in the following way (as illustrated in Figure 2 (5,6)):

- Given a desired map size $h \times w$, and a set of trained MdMCs as described above.
- A high-level map is sampled using the high-level chain.
- The low-level map is sampled using the low-level chains. To sample a given tile in the low-level map, we check the corresponding high-level tile in the high-level map, which determines which of the low-level chains to use. In the example shown in Figure 2 (5,6), the current low-level tile will be sampled using the low-level chain trained for the high-level tile, "h_1."

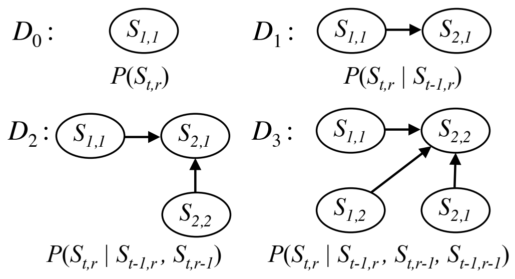


Figure 3: The dependency structures used in our experiments. Notice that D_0 is a histogram, D_1 is a standard Markov chain, and D_2 and D_3 are examples of MdMCs.

While sampling, our method may encounter a set of previous tiles that was not encountered during training. The probability estimation for this configuration would thus have not been properly estimated (absolute counts would be 0). We call this an *unseen state*. We assume unseen states are undesirable, because many correspond to ill-formed structures in our application domains. We incorporate two strategies from our previous work (Snodgrass and Ontanon 2014b) to help us avoid unseen states: *look-ahead* and *fallback*. These methods attempt to sample with the condition that no unseen states are reached. The look-ahead process attempts to sample a fixed number of tiles in advance, making sure that no unseen state is reached. If the look-ahead is unsuccessful, and a tile cannot be found that results in no unseen states, then our method falls back to a simpler (i.e., lower order) dependency structure. More details of how this process works can be found in (Snodgrass and Ontanon 2014b).

Experiments

We chose to use the platformer game *Super Mario Bros.* and the puzzle game *Loderunner* as our application domains[3]. Both games feature two-dimensional tile-based maps, but *Super Mario Bros.* maps are linear (in the sense explored by Dahlskog et. al. (2014)), while *Loderunner* maps are highly non-linear, and contain many maze-like characteristics.

Super MarioBros.: For our experiments, we used 12 outdoor *Super Mario Bros.* maps to train our models. We represent each *Super Mario Bros.* map using a set of 7 low-level tiles. **S** is a sentinel tile referring to the outer boundary of a map. The remaining tiles correspond to components of the maps: **E** represents empty space, **B** is a breakable block, **?** is a ?-block, **p** represents the left portion of a pipe, **P** represents the right portion of a pipe, and **G** is the ground.

Loderunner: This is a puzzle game where the player must collect treasure while avoiding guards. The player is able to dig holes to trap guards and reach different areas. For our experiments we used 150 *Loderunner* maps to train our models. We represent each *Loderunner* map using a set of 9 low-level tiles. **S** is a sentinel tile. The remaining tiles are defined as follows: **E** represents empty space, **b** and **B** are sections of ground that can and cannot be dug by the player, respectively, **H** is a ladder, **T** is treasure, **G** is a guard, **R** is a rope, and **P** represents the player's starting point.

Experimental Setup

We applied our method to the two games above, experimentally varying the following parameters:

- **Number of Clusters** (k): map chunks are clustered into k groups, and we use the medoids of those groups as the high-level tiles. Therefore, k gives the number of high-level tiles. We experimented with $k \in \{6, ..., 16\}$.
- **Distance Metric** (C): C is used to determine the distances between different level chunks for clustering, and translate the input maps into a high-level tile representation. We experimented with $C \in \{Direct, Histogram, Markov, Shape, WeightedDirect\}$.
- **Tile Size** (T): high-level tiles are $T \times T$ chunks of low-level tiles. We experimented with $T \in \{3, 4, 6\}$.

To test these configurations, we chose a baseline configuration of $T = 4$, $k = 15$, and $C = WeightedDirect$ based on observed performance over some preliminary tests. A section of a map sampled using the baseline configuration can be seen in Figure 4. We compare against other configurations where only one of the above variables is changed from the baseline. For all configurations, we use D_2 as the high-level dependency structure, which falls back to D_1, and then to D_0. We use D_3 as the low-level dependency structure, which falls back to D_2, then to D_1, and then to D_0. We evaluated the results using the following metrics:

- **Playability**: For *Super Mario Bros.*, we loaded the maps into the 2009 Mario AI competition software (Togelius, Karakovskiy, and Baumgarten 2010) and had Baumgarten's A^* agent play them. For *Loderunner*, we check for the existence of a path between all treasures (collecting all treasures completes the level). For each configuration we recorded the percentage of playable maps. Notice, that if our methods were to be used for a released game, we could easily set up a failsafe that simply rejects any unplayable map, and samples a new map in its place. However, for the sake of understanding the performance of our methods, we don't use such failsafe in this paper.
- **Linearity**: This metric is used to evaluate the *Super Mario Bros.* maps. Linearity refers to the general trend of the vertical positions of platforms and ground in the level (Smith and Whitehead 2010). Linearity is measured by treating each platform and ground section as a point and using linear regression to find a best-fit line for those points. The sum of distances between each point and the line is normalized into [0,1], giving us the linearity value for that map. The linearity value is inversely proportionate to how linear a map is.
- **Leniency**: This metric is used to evaluate the *Super Mario Bros.* maps. Leniency corresponds to how forgiving a map is (Smith and Whitehead 2010). That is, how likely a player is to be harmed. Leniency is measured by counting the number of places in the map where a player can die. We did not annotate enemies in our training maps, and thus, our method does not place enemies, therefore, we only count the gaps which result in death if not cleared. We weight each of these gaps by its length.

Playability is measured using 100 sampled maps in *Super Mario Bros.* with each configuration, and 50 maps in each configuration for *Loderunner*. Linearity and leniency are computed for 1000 maps with each configuration. We compare the linearity and leniency with those obtained by the top three competitors from the level generation track of the 2011 Mario AI competition: Takahashi, Baumgarten, and Mawhorter (Mawhorter and Mateas 2010)(generating 1000 maps with each of them). For *Super Mario Bros.*, we sampled maps that were 12×210, and 32×28 *Loderunner* maps.

[3]*Super Mario Bros.* and *Loderunner* training sets are available for download from the author's website at: https://sites.google.com/site/sampsnodgrass

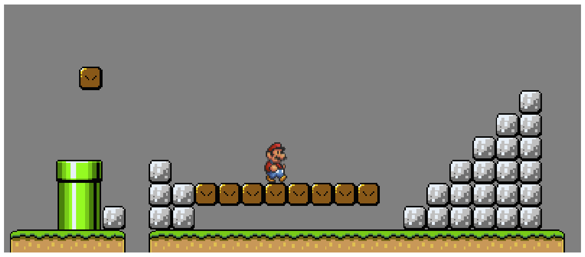


Figure 4: A section of a *Super Mario Bros.* map sampled using the *WeightedDirect* metric, $k = 15$, and $T = 4$.

Results

In this section we discuss the results obtained by varying the number of clusters, distance metric, and tile size. The baseline of our approach is marked with an asterisk.

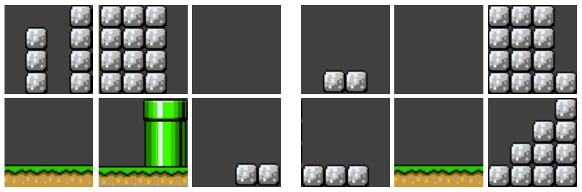


Figure 5: An example set of high-level tiles extracted using $C = Histogram$ (left) and $C = WeightedDirect$ (right), with $k = 6$, and $T = 4$.

Table 1: Playability

	Super Mario Bros.	*Loderunner*
Histogram	38%	22%
Direct	44%	10%
Markov	60%	16%
Shape	44%	26%
*Weighted**	66%	28%
3×3	44%	22%
4×4*	66%	28%
6×6	28%	18%
Non-Hier	49%	34%
Auto-Hier*	66%	28%
Manual-Hier	80%	-
Mawhorter	94%	-
Takahashi	84%	-
Baumgarten	2%	-

Number of Clusters. Figure 6 shows the percentage of maps sampled that are playable for *Super Mario Bros.* and for *Loderunner*, using the playability metrics described previously. Notice that as the number of clusters increases, there is a general increase in the percentage of playable *Super Mario Bros.* maps, whereas the percentage of playable *Loderunner* maps remains near 20%. *Super Mario Bros.* is not a structurally complex game, that is, many of the chunks found in the training set are similar, if not identical. Alternatively, *Loderunner* has a much larger variety of chunks, due in part to a larger training set, but also because *Loderunner* maps rely more on maze-like structures and puzzles in the level design. Therefore, we hypothesize that when learning a model for *Loderunner* a much larger number of clusters may be needed to model the space.

Distance Metric. The top section of Table 1 shows the effect of changing the distance metric. Notice that when using the *Histogram* distance metric for *Super Mario Bros.*, we achieve the lowest percentage of playable maps. This is because there are many structures composed of just a few tile types, and the *Histogram* is unable to create an appropriate cluster for each structure. Notice also, that the *Weighted-Direct* metric achieves a high percentage of playable maps (66%). This is due to the metric being able to cluster similar structures together. Figure 5 shows the medoids generated by our system using the *Histogram* and *WeightedDirect* distance measures with $k = 6$. For *Loderunner*, the *Direct* and the *Markov* distance metric both perform worse than the other metrics. The *Direct* metric is unable to create the

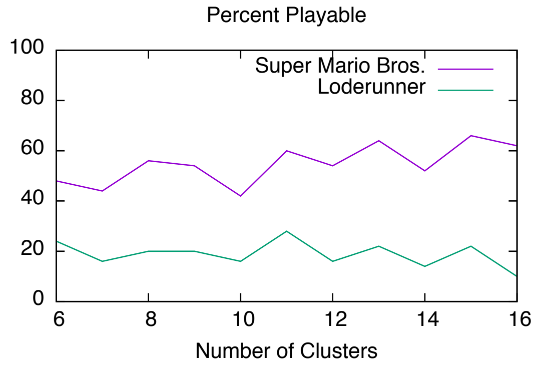


Figure 6: A graph showing the percentage of playable sampled maps, for *Super Mario Bros.* and *Loderunner*, as a function of the number of clusters.

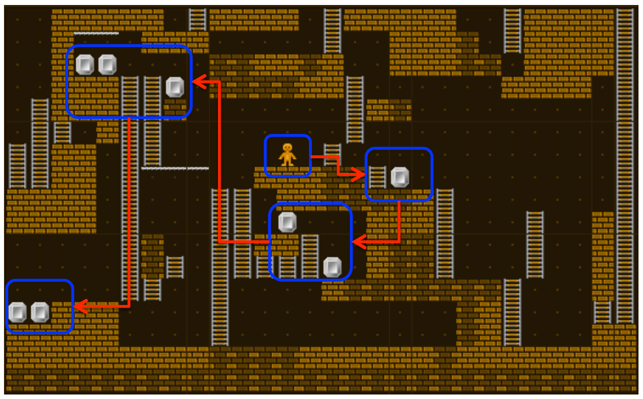


Figure 7: A Loderunner map sampled using the the non-hierarchical approach. The blue areas denote the starting area and treasure areas. The arrows indicate a possible solution path between the areas.

proper clusters because there are too many different chunks for meaningful structures to be extracted using a direct tile by tile comparison. The *Markov* metric performs poorly because the maps are not sequential.

Tile Size. The second section of Table 1 shows the effect of varying the size of the high-level tiles. In terms of playability we can see that 4×4 tiles achieve the best results in both domains. Complete high-level structures are difficult to capture in a small 3×3 tile. In *Super Mario Bros.* this can be seen empirically. When using the 3×3 tile, maps are often sampled that have multiple high-level tiles corresponding to a pipe stacked on top of one another. This leads to structures that are too tall to pass. Using a 6×6 tile exponentially increases the possible number of tile configurations that can be seen, and therefore the number of high-level structures that need to be found. Without an equal increase in the training data, we are unable to find all such structures, leading to clusters that do not properly model the space.

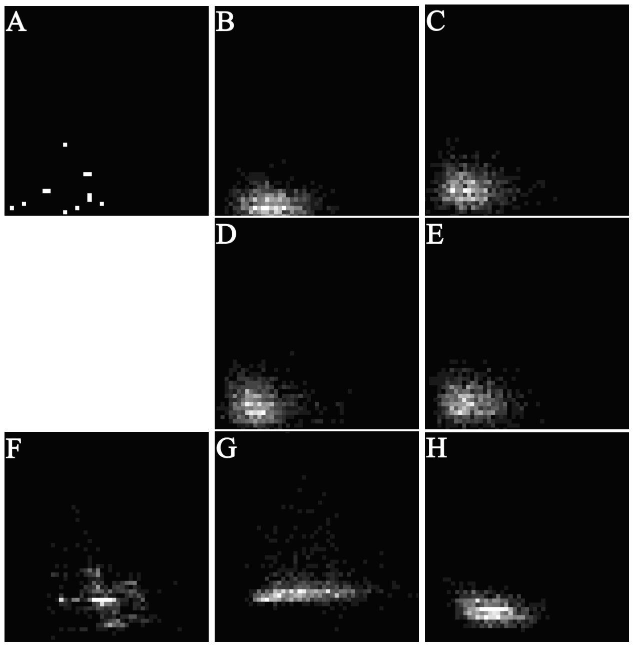


Figure 8: Plots of the expressive range of each generator obtained by plotting leniency against linearity for 1000 maps for each generator. The brightness of a point represents the number of maps sampled with those linearity (horizontal axis) and leniency (vertical axis) values. A) training maps; B) non-hierarchical method; C) manual hierarchical method; D) and E) automatic hierarchical method using the $Markov$ and $WeightedDirect$ distance metrics, respectively; F) Baumgarten's; G) Mawhorter's; and H) Takahashi's.

Comparison. The last section of Table 1 shows a comparison between the best results using a non-hierarchical approach, the best results using the approach outlined in this paper (achieved using the *WeightedDirect* distance metric, $k = 15$, and $T = 4$), the best results obtained using hand crafted high-level tiles, and the results using the three top performers of the 2011 Mario AI competition level generation track (Shaker et al. 2011). It is important to note that the difference in the playability results of our approach (66%) applied to *Super Mario Bros.* is statistically significant as compared to results of our non-hierarchical approach (49%). Additionally, the difference in playability between our automatic hierarchical approach and our manual hierarchical approach (80%) is not statistically significant (both tested using a one-tailed t-test with $p = 0.1$). That is to say, our approach performs significantly better than the non-hierarchical approach, and statistically the same as the manual hierarchical approach for *Super Mario Bros.* Further, notice that our approaches do not sample as many playable maps as the Mario AI competitors, excluding Baumgarten, whose generator consistently placed large sections of breakable blocks that the agent was unable to navigate (which doesn't necessarily mean the maps are not playable). However, our approach is general and applicable to multiple domains, whereas the Mario AI competitors are specialized to *Super Mario Bros.* and cannot be used in other domains without significant changes. Notice, the playability results achieved with our approach for *Loderunner* and those of the non-hierarchical approach are not statistically significant (tested using a one-tailed t-test with $p = 0.1$). This may be because *Loderunner* maps are not structured sequentially in a way our MdMC model can model. Figure 7 shows a map sampled using the non-hierarchical approach, and one potential solution path to that map.

Expressive Range. The expressive range refers to the variety of maps a generator is able to produce. In our experiments we measure the expressive range of a generator by plotting each map sampled, using linearity and leniency as the axes. Figure 8 shows the expressive ranges of our method using the $Markov$ (D) and $WeightedDirect$ (E) distance metrics, our method using the manually defined high-level tiles (C), our non-hierarchical approach (B), the top three competitors of the Mario AI competition (F, G, H), and the original twelve *Super Mario Bros.* maps that our methods use for training (A). Notice that the expressive ranges of our approaches roughly match that of the training set, which is to be expected. The expressive range of Mawhorter's generator (center) has a more constrained leniency than the other generators. Mawhorter's generator creates difficulty in its levels more by placing enemies rather than by placing holes. Our evaluations ignore enemies, making these levels appear to be more lenient. Baumgarten's generator places large pre-authored sections together, which explains its fragmented expressive range. Lastly, Takahashi's generator's expressive range looks similar to our approaches' expressive ranges, albeit shifted and scaled slightly.

Conclusions

This paper presented a hierarchical approach to generating video game maps using multi-dimensional Markov chains (MdMCs) for learning and sampling, and clustering for finding high-level structures. Our approach explores the concept of controlling a content generator with training data instead of domain knowledge. We found that a hierarchy of MdMCs provides a significant improvement over a similar non-hierarchical approach without a significant loss as compared to the manual hierarchical approach. We want to investigate whether using Markov random fields in place of MdMCs can provide further improvements, and at what cost. Markov random fields would be especially interesting in games such as *Loderunner*, where the maps are not linear, but exhibit complex maze-like structures. We are also interested in determining how much the training data shapes the expressive range of our approach, and how the amount of training data might affect the types of maps sampled.

Acknowledgments

We would like to thank Michael Ashmore for collecting the Loderunner dataset.

References

Ching, W.-K.; Huang, X.; Ng, M. K.; and Siu, T.-K. 2013. Higher-order markov chains. In *Markov Chains*. Springer. 141–176.

Conklin, D. 2003. Music generation from statistical models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, 30–35. Citeseer.

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

Hendrikx, M.; Meijer, S.; Van Der Velden, J.; and Iosup, A. 2013. Procedural content generation for games: a survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)* 9(1):1.

Kaufman, L., and Rousseeuw, P. 1987. *Clustering by means of medoids*. North-Holland.

Levina, E., and Bickel, P. J. 2006. Texture synthesis and nonparametric resampling of random fields. *The Annals of Statistics* 1751–1773.

Markov, A. 1971. Extension of the limit theorems of probability theory to a sum of variables connected in a chain.

Mawhorter, P., and Mateas, M. 2010. Procedural level generation using occupancy-regulated extension. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 351–358. IEEE.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P.; Takahashi, G.; et al. 2011. The 2010 mario AI championship: Level generation track. *TCIAIG, IEEE Transactions on* 3(4):332–347.

Shannon, C. E. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5(1):3–55.

Smith, A. M., and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on* 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*, 4. ACM.

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

Snodgrass, S., and Ontanon, S. 2014b. A hierarchical approach to generating maps using markov chains. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

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