The AIIDE-17 Workshop on
Experimental AI in Games
WS-17-19

Leveraging Multi-Layer Level Representations for Puzzle-Platformer Level Generation

Sam Snodgrass, Santiago Ontañón
Drexel University
3141 Chestnut St
Philadelphia, PA 19104
sps74@drexel.edu, so367@drexel.edu

Abstract

Procedural content generation via machine learning (PCGML) has been growing in recent years. However, many PCGML approaches are only explored in the context of linear platforming games, and focused on modeling structural level information. Previously, we developed a multi-layer level representation, where each layer is designed to capture specific level information. In this paper, we apply our multi-layer approach to *Lode Runner*, a game with non-linear paths and complex actions. We test our approach by generating levels for *Lode Runner* with a constrained multi-dimensional Markov chain (MdMC) approach that ensures playability and a standard MdMC sampling approach. We compare the levels sampled when using multi-layer representation against those sampled using the single-layer representation; we compare using both the constrained sampling algorithm and the standard sampling algorithm.

Introduction

Procedural content generation (PCG) studies the algorithmic creation of content (e.g., levels, textures, items, etc.), often for video-games. PCG via machine learning (PCGML) (Summerville et al. 2017) is the use of machine learning techniques to create a model from which to sample content. Recently there has been increased interest in PCGML for video game levels (Guzdial and Riedl 2016; Dahlskog, Togelius, and Nelson 2014; Summerville and Mateas 2016; Snodgrass and Ontañón 2016b). However, in most cases PCGML techniques have been used to generate linear platformer levels (e.g., *Super Mario Bros.* levels), and typically only capture structural level information. We previously developed a multi-layer representation in order to capture more varied information from the training levels (Snodgrass and Ontañón 2017). We tested this approach in the domain of linear platformer level generation, but we believe this approach can be used to model and generate levels in more complex domains, because of its ability to more deeply represent domains. Therefore, we explore the use of this multi-layer technique in *Lode Runner*, a puzzle-platformer game that requires complex paths to complete levels, and allows for level destruction to open up more paths.

The remainder of the paper is organized as follows. We first formulate the specific problem we are trying to address. We then give background on recent PCGML techniques. Next, we discuss our multi-layer representation, multi-dimensional Markov chain approach, and our playability constraint for non-linear paths. We then describe our experimental set-up and present our results. We close by drawing our conclusions and suggesting future work.

Problem Statement

In this paper we address the problem of generating playable levels in domains that require complex paths. We do this through the use of a multi-layer level representation and modeling approach and a constrained sampling algorithm. Specifically, we model the game *Lode Runner* using a structural layer, a player path layer, and a section layer in order to more accurately capture the intricacies of the domain.

Related Work

Procedural content generation via machine learning (PCGML) is the algorithmic creation of content using machine learning models trained on some form of training data (Summerville et al. 2017). This section discusses PCGML approaches.

There are several PCGML approaches for level generation; we discuss a few recent approaches below. Summerville and Mateas (2016) employed long short-term memory recurrent neural networks to generate *Super Mario Bros.* levels accounting for player paths via level annotations. Guzdial and Reidl (2016) used Bayesian networks to generate levels, also for *Super Mario Bros.* implicitly accounting for player interactions through model parameters approximated by observing player movements in videos. Notice, that though these methods model more than just the structural information of the level, the way the information is included is not easily extensible to other types of information. There are other PCGML level generation approaches (Dahlskog, Togelius, and Nelson 2014; Hoover, Togelius, and Yannakis 2015; Shaker and Abou-Zleikha 2014), but notice that the majority of approaches have only been tested in the domain of linear platformer level generation.

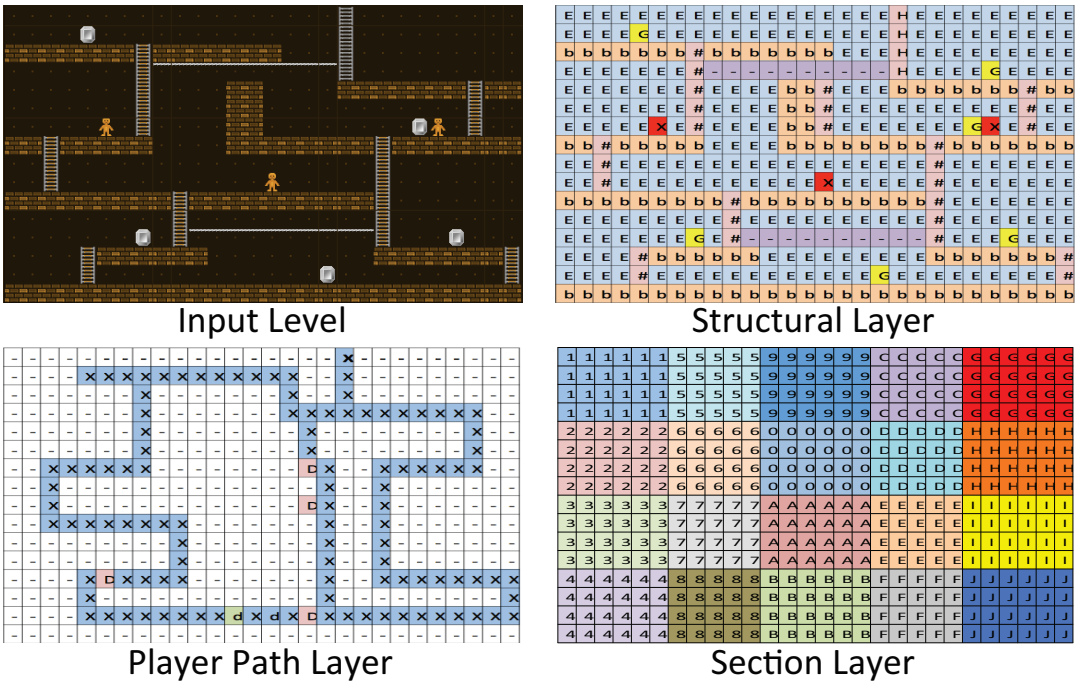

Figure 1: This figure shows a *Lode Runner* level (top-left) represented using a structural layer (top-right), a player path layer (bottom-left), and a section layer (bottom-right). Color is added for clarity.

Methods

In addition to the approaches above, there are a few approaches that leverage multiple layers of representation. We previously explored a hierarchical approach that used clustering to find common structures in levels (Snodgrass and Ontanon 2015). The domain was then modeled at multiple levels, and new levels were sampled at increasing resolution. Notably, we applied this approach to *Lode Runner* with little success (i.e. sampling very few usable levels). Summerville and Matteas (2015) explored a hierarchical approach to dungeon generation, by modeling the layout of the dungeon and the structure of the rooms separately. Notice, however, that both of these approaches focus on only the structural elements of the levels.

To more faithfully and deeply represent and model levels, we recently developed a multi-layer approach that is able to capture structural information, and other level information (such as player paths) in a general format (Snodgrass and Ontañón 2017). We previously used this approach to generate linear platformer levels, but in this paper we show that it is usable in more complex domains, such as *Lode Runner*.

In this section we discuss our multi-layer representation introduced in (Snodgrass and Ontañón 2017), explain how we train and sample from an MdMC with a multi-layer representations, and introduce a playability constraint.

Multi-layer Level Representation

We represent a level using a set of layers, $L = \{l_1, l_2, ..., l_n\}$, where each l_i is a two-dimensional matrix of tiles with dimensions $h \times w$. Each layer has a separate set of tile types, T_i, with meanings defined by the layer.

Figure 1 shows a *Lode Runner* level (top-left) represented using three layers: a *structural* layer (top-right), representing the placement of objects in the level; a *player path* layer (bottom-left), representing the path a player may take through the level; and a *section* layer (bottom-right), representing different sections of the level;. We refer to the structural layer as the *main* layer, because it is the layer that provides the tile types we will use during level sampling.

Though this is a straightforward concept, it opens up the possibility for more faithful and deep level representation than is possible with previous representations which only capture structural information (Guzdial and Riedl 2016; Snodgrass and Ontañón 2016b) and occasionally player path information (Summerville et al. 2016a). For example, in this paper in addition to a structural and a player path layer, we use a section layer which signifies different sections of the level. Notice that these are only two possible additional layers. Others can include a difficulty layer, which could capture the progression of difficulty as you progress through it, or an enemy path layer which captures the behavior of the enemies.

Markov Chain-based Level Generation

We now explain how we train and sample from our model. We first introduce multi-dimensional Markov chains, before discussing our training and sampling approaches.

Markov Chains Markov chains (Markov 1971) model stochastic transitions between states over time. A Markov chain is defined as a set of states, $S = \{s_1, s_2, ..., s_n\}$, and

the conditional probability distribution (CPD), $P(S_x|S_{x-1})$, representing the probability of transitioning to a state $S_x \in S$ given the previous state, $S_{x-1} \in S$. The set of previous states that influence the CPD are the *network structure*.

Multi-dimensional Markov chains (MdMCs) are an extension of higher-order Markov chains (Ching et al. 2013) that allow any surrounding state in a multi-dimensional graph to be considered a previous state. In our case, the multi-dimensional graph includes previous tiles in the current layer as well as from other layers. For example, the CPD defined by nsl_3 in Figure 2 (right) can be written as $P(S_{x,y}|S_{x-1,y}, S_{x,y-1}, S_{x-1,y-1}, Q_{x,y}, R_{x,y})$, where S, Q, and R the set of states in the various layers. Notice, that the set of states are also the set of tile types for each of the layers. Further, in this case S is the set of states (and tile types) for the main layer (i.e. the layer we wish to generate through sampling). By redefining what a previous state can be in this way, the model is able to more easily capture relations from multi-dimensional training data, as shown in our previous work (Snodgrass and Ontañón 2016b).

Training In this section we discuss how we estimate a conditional probability distribution (CPD) with a single-layer multi-dimensional Markov chain (MdMC), and then how we estimate a CPD with our multi-layer MdMC.

Training a single-layer MdMC requires: 1) the network structure and 2) training levels. The network structure specifies which of the surrounding states the value of the current state depends upon. This set of surrounding states and their values is referred to as the "previous tile configuration" or "previous configuration." Using the network structure and training levels, the conditional probability distribution, P, of each tile given each previous configuration is calculated according to the frequencies observed in the training data.

Training a multi-layer MdMC requires a network structure, and training levels represented in multiple layers. The conditional probability distribution, P_m, is computed much the same way as for the single-layer approach. The main difference between training a single-layer and multi-layer MdMC is that the network structure of the multi-layer MdMC may contain states from other layers, allowing P_m to learn dependences across multiple layers of representation. Figure 2 shows network structures that can be used to train a single-layer MdMC (left) and a multi-layer MdMC (right).

Sampling In this section we describe how we sample new levels: first, using a single-layer MdMC; then a multi-layer MdMC, and finally a constrained sampling extension.

To sample a new level using a single-layer MdMC, we need desired level dimensions, $h \times w$, and the conditional probability distribution, P, trained as above. The new level is then sampled one tile at a time starting, for example, in the bottom left corner and completing an entire row before moving onto the next row. For each position in the level, a tile is sampled according to P and the previous configuration. While sampling, we use a *look-ahead* and *fallback* procedure. The look-ahead procedure works as follows: when sampling a given tile, the process generates a number of tiles ahead to make sure that with the tile that has been selected, we will not reach an *unseen state* (a combination of tiles

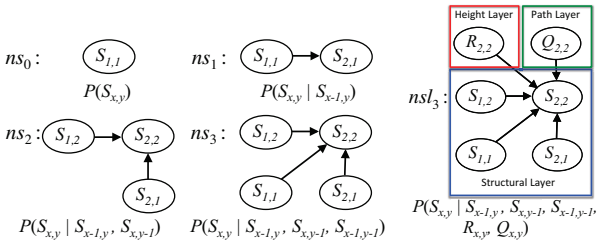

Figure 2: This figure shows the network structures used by the single-layer MdMC (left), and a network structure used by the multi-layer MdMC. Note, that each nsl_i follows the same pattern as the corresponding ns_i, but the multi-layer network structures retain the dependencies from the other layers in each network structure.

that was not observed during training, and that we, thus, do not have a probability estimation for). If the sampled tile will result in an unseen state, then a different tile is sampled. The fallback procedure comes into play when unseen states are unavoidable according to the looklook-aheadahead procedure. Instead of training a single MdMC model with a single network structure, we train a collection of MdMCs (with increasingly simple network structures). When sampling a new tile, the most complex model is used first, and if it cannot generate a tile satisfying the look-ahead, then we fallback to a simpler one. For the single layer model, network structure ns_3 falls back to ns_2, then to ns_1, and finally ns_0, which is the raw distribution of tiles observed during training. Notice, this fallback approach is analogous to the backoff models used for n-grams (Katz 1987). More information on the look-ahead and fallback procedures as they apply to MdMCs can be found in (Snodgrass and Ontañón 2016b).

Sampling a level using a multi-layer MdMC functions the same as the single-layer method, but with one adjustment: because the trained distribution, P_m, only models the probability of tiles in the main layer, and previous configurations contain states from the other layers, the other layers must be defined before sampling or computed during. We define the non-structural layers prior to sampling. Notice, Figure 2 (right) only shows nsl_3 for the multi-layer model. However, the multi-layer network structures follow the same pattern as the single-layer network structures, but retain the dependencies from the other layers. For example, nsl_2 would depend on the tiles to the left and below in the structural layer (as in ns_2), as well as on the tiles at the current position in the section and player path layers. The fallback order for the multi-layer and single-layer models are the same.

Lastly, in order to ensure playable and well-formed levels, we employ an extension to the above sampling approach, which accepts constraints (such as ensuring playability, or forcing the number of enemies to be in a certain range) and enforces them through a resampling process. Below we discuss the constrained sampling algorithm in more detail.

Algorithm 1 shows the *Violation Location Resampling* or *VLR* algorithm(Snodgrass and Ontañón 2016a). This algorithm takes the desired dimensions of the output map and

Algorithm 1 ViolationLocationResampling(w, h, C)

```
1:  Map = MdMC([0, 0], [w, h])
2:  while (Σ_{c∈C} c(Map).cost) > 0 do
3:    for all c ∈ C do
4:      for all ([x1, y1], [x2, y2]) ∈ c(Map).sections do
5:        for all c_i ∈ C do
6:          cost_{c_i} = c_i(Map[x1, y1][x2, y2]).cost
7:        end for
8:        repeat
9:          m = MdMC([x1, y1], [x2, y2])
10:         for all c_i ∈ C \ c do
11:           if cost_{c_i} > c_i(m).cost then
12:             GoTo line 9
13:           end if
14:         end for
15:       until c(m).cost < cost_c
16:       Map[x1, y1][x2, y2] = m
17:     end for
18:   end for
19: end while
20: return Map
```

a set of constraints, C, and returns a map satisfying those constraints. Each constraint returns a cost associated with the map and sections of the map that can be resampled to reduce that cost. The algorithm begins by sampling a new main layer, Map, using the multi-layer sampling approach described above (line 1). Next, if any constraints return a nonzero cost in the main layer (line 2), then for each constraint, $c \in C$ (line 3), the algorithm iterates over the sections of the main layer which have a nonzero cost (line 4). It then records the cost of the current section according to each constraint (lines 5-7). The algorithm then samples a new section, m, of the same dimensions as the current section (line 9), until the cost of m is lower than the previous cost for c, and it does not increase the cost of any other constraint (lines 8-15). This process of finding violated sections and improving their costs is repeated until the total cost of Map is 0 (line 2).

Playability Constraint

Lode Runner is a puzzle-platforming game that often requires complex paths through levels, and allows actions that temporarily change the level geometry which may open additional paths. Because *Lode Runner* allows for more complex actions and requires non-linear paths, determining where the playability of a level breaks down is more difficult. Notice, in linear games, such as *Super Mario Bros.*, we can simply determine how far into a level an agent is able to reach, and then resample that section until it is passable; in *Lode Runner* determining the appropriate sections to resample is not as straightforward. We can easily determine whether a level is playable by checking that a path exists that passes through each gold, however, in addition to simply checking whether a level is playable, we also need to be able determine where in the level playability breaks down. We frame this playability problem as one of adding edges to a graph in order create strongly connected graphs, as we explain below.

We determine the sections to resample in several stages:

1. Select positions in the level for which passage is important. We selected positions containing gold (which must be collected to complete the level), and boundaries between sections, as defined by the section layer.
2. Create a graph where each node is a position selected above. Connect two nodes with a directed edge if a path exists between the positions in the level that does not pass through the other positions. We determine if such a path exists with a specialized *Lode Runner* agent.
3. Find the strongly connected components of the graph with, for example, Tarjan's algorithm (Tarjan 1972).
4. Create a directed acyclic graph (DAG) treating each strongly connected component as a node, and adding edges between nodes in the DAG when the strongly connected components have edges between them.
5. If all of the gold pieces are within one strongly connected component, then the level is playable, and no sections need to be resampled. Otherwise, continue with the remainder of the steps.
6. Find the sets of nodes in the DAG that have an in-degree of 1, called X, or an out-degree of 1, called Y.
7. If $|X| > |Y|$, return for resampling all sections which: a) are a part of the strongly connected component represented by the node in X, and b) contain a gold.

By resampling sections with gold that are bottlenecks in the graph, we are able to increase traversability through the level, thus making the gold more easily reachable.

Notice, that this approach can be used to check for paths between arbitrary points in a level. The only domain-specific element is the agent which finds paths between positions, but this can be replaced with any agent that is able to determine if one position is reachable from another in some domain.

Experimental Evaluation

We test our approach by sampling levels for the classic video game, *Lode Runner*. Our goal is to determine whether the multi-layer approach is able to more reliably and easily generate playable levels than the single-layered approach, and if providing a play path layer allows for more control over the output levels. Additionally, we evaluate whether the violation location resampling algorithm is necessary when using the multi-layered approach. The remainder of the section discusses our chosen domain in more detail, describes the experimental set-up, and reports the results of our experiments.

Domain

Lode Runner is a puzzle-platforming game which requires complex non-linear paths, and lets the player modify the structure of the levels. To complete a level in this domain, the player must collect all of the *gold* pieces placed in the level. In our experiments we use 10 levels from *Lode Runner*

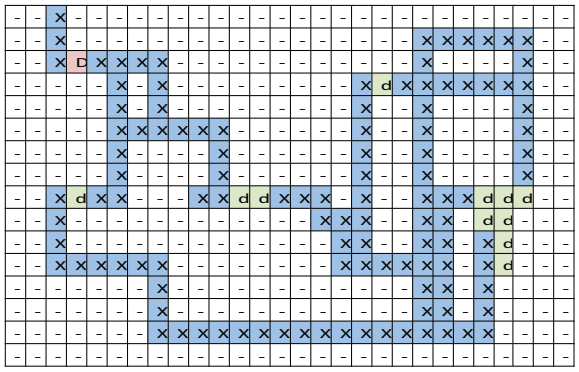

Figure 3: This figure shows the player path layer used with the levels generated with "Multi P6."

on the NES. These levels could be found in the video-game level corpus (Summerville et al. 2016b).

Level Representation

In this section we describe the layers of our training levels.

Structural Layer The structural layer captures the placement of objects, enemies, and items throughout the level. This layer is represented by an array, where each cell takes its value from a set of 10 tile types representing elements in the level, such as ladders, gold, enemies, etc. Figure 1 (top-right) shows the structural layer of a *Lode Runner* level.

Section Layer The section layer captures positioning information about the level. This layer essentially splits the level into several sections to allow for more focused training within each of the sections. For our experiments, we use a set of 20 tiles to represent the section layer. We split each level into 5×4 and 6×4 (width $\times$ height) tile sections, depending on the positioning within the level. We chose these sizes based on preliminary experiments. Figure 1 (bottom-right) shows the section layer of a *Lode Runner* level.

Player Path Layer The player path layer captures the path a player took through the level. This layer is represented with 4 tile types: $-$ represents a position the player did not pass through; d represents a position where the player performed the "dig left" action, temporarily destroying a piece of breakable ground to the left and below the player; D represents a position where the player performed the "dig right" action; X represents a spot the player moved through without taking either dig action. We extracted these paths from a video of a player completing the levels[1]. Figure 1 (bottom-left) shows the player path layer of a *Lode Runner* level.

Experimental Set-up

In order to explore the effectiveness of modeling our chosen domain with the multi-layered approach, we train 2 models: a multi-layer model using all three layers described previously, and a single-layer model using only the structural layer. The single and multi-layer models require several parameters to be set before training and sampling.

- **Single-layer MdMC**: This requires setting a look-ahead, and a set of network structures to be used during training, sampling, and for the fallback. In our experiments, we use a look-ahead of 3. For the network structures, we use ns_3 as the main network structure. During sampling, ns_3 falls back to ns_2 which falls back to ns_1 which falls back to ns_0. Network structures can be seen in Figure 2 (left).
- **Multi-layer MdMC**: This requires setting a look-ahead, and a set of network structures to be used during training, sampling, and for the fallback. Any layer not being sampled (i.e. non-structural layers) must be provided at the time of sampling or computed during; we provide the non-structural layers prior to sampling. For our experiments, we use a look-ahead of 3. For the network structures, we use nsl_3 as the main network structure. During sampling, nsl_3 will fall back to nsl_2 which will fall back to nsl_1 which finally falls back to nsl_0. Recall, the nsl_i network structures follow the same pattern as the ns_i network structures, but always retain dependencies on the other layers. During sampling, we provide the section layer from Figure 1 (bottom-right), and sample 50 levels with each player path layer from the training levels.

We use each trained model paired with the violation location resampling algorithm as well as the standard MdMC sampling algorithm. For the VLR algorithm, we enforced the playability constraint as described previously.

We sampled 500 levels with the single-layer and multi-layer model paired with each sampling approach (VLR and standard sampling). We are interested in determining whether the multi-layer approach allows us to more easily sample usable levels for complex domains. Therefore, we record the playability of all levels sampled using the standard sampling approach. Additionally, we compare how easily the multi-layered approach generated playable levels as compared to the single-layer approach when using the VLR algorithm. We determine this by setting a limit on the number of sections that can be resampled (100 sections), and recording how many levels are unfinished as well as the average number of sections resampled per finshed level for each model. Lastly, we are interested in whether the multi-layer approach gives the user more control over the sampled levels. This is determined by computing the percentage of gold pieces placed in the level that appear on the provided player path. We compare this percentage for the multi-layer approach against the average over all player paths for the single-layer approach as a baseline.

Results

Table 1 shows the results of sampling levels with both the multi-layer and single-layer models paired with the violation location resampling algorithm. Each "Multi" represents the 50 levels sampled using the path layer from one of the training levels. The first column (Average) is the average number of sections resampled counting only the sections resampled in levels that did not hit the limit of resampled sections. The second column (Std. Dev.) is the standard deviation of

[1] www.youtube.com/watch?v=VwxLChxi8WA

Table 1: Multi-layer vs. Single-layer VLR Comparison

	Sections Resampled		
Model	Average	Std. Dev.	Unfinished
Multi P0	14.250	21.289	12%
Multi P1	23.267	27.522	10%
Multi P2	33.028	31.658	28%
Multi P3	20.281	25.213	36%
Multi P4	20.629	20.550	30%
Multi P5	45.677	27.849	38%
Multi P6	35.118	29.555	66%
Multi P7	30.452	30.811	38%
Multi P8	38.882	33.689	32%
Multi P9	13.625	19.377	4%
Multi All	18.334	18.474	29.4%
Single	17.359	20.712	27.0%

the number of sections resampled, again only counting completed levels. The final column (Unfinished) shows the percentage of levels that were abandoned during sampling because they resampled too many sections. Note, the values for "Multi All" are the weighted averages based on the number of completed levels for each path layer.

We see in Table 1 that the single and multi-layer approaches are both able to reliably generate playable levels, on average, using the VLR algorithm. In fact, there is no statistically significant difference between the average number of sections resampled (when $p = 0.1$, using a T-test), or between the average number of unfinished levels (when $p = 0.1$ with a chi-square test). However, despite their similar performance on average, notice that the choice of path can have a large impact on the VLR's ability to sample levels, as can be seen in the "Multi P6" row of the table. Figure 3 shows the "Multi P6" path. Notice it contains several groups of movements and dig actions together in the path than the path shown in Figure 1 (bottom-right), which is the "Multi P0" path, which makes it more difficult to replicate.

Figure 4 shows a level sampled using the multi-layer approach (top) and with the single-layer approach (bottom), both sampled with the VLR algorithm. We annotated a possible solution to each level. Note that red *X*'s represent sections of ground that need to be destroyed by the player to complete the level.

Table 2 shows the percentage of playable levels sampled with both the multi-layer and single-layer models paired with the standard sampling approach. As above, each "Multi" represents the 50 levels sampled using the path layer from one of the training levels, and "Multi All" represents the percentage playable over all the levels sampled with the multi-layer model. On average, we see the single-layer approach is able to sample significantly more playable levels (with $p = 0.1$, using a chi-square test). However, we see the performance of the multi-layer approach heavily depends upon the player path layer used during sampling. For instance, when a path includes a lot of digging actions and clumped sections of movements (as in "Multi P6" in Figure 3) the model has a more difficult time capturing the structural patterns surrounding the movements, and thus creates

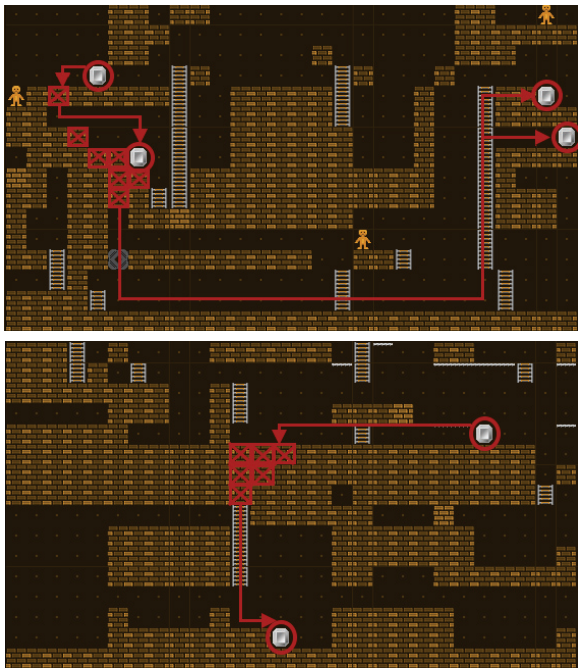

Figure 4: This figure shows a level sampled using the multi-layer approach (top) and a level sampled using the single-layer approach (bottom), both with the VLR algorithm. The levels have been annotated with solutions, where red *X*'s indicate ground that must be destroyed.

Table 2: Multi-layer vs. Single-layer Standard Comparison

Model	Percent Playable
Multi P0	30%
Multi P1	10%
Multi P2	18%
Multi P3	10%
Multi P4	20%
Multi P5	4%
Multi P6	6%
Multi P7	4%
Multi P8	6%
Multi P9	24%
Multi All	13.2%
Single	32.6%

fewer playable levels.

Table 3 shows the percentage of golds in the sampled levels that are placed on the provided player path. We believe this metric gives an approximation of how well the sampled levels adhere to the provided path, and therefore how much control providing a path gives the user. For the single-layer approaches we computed the average percentage of gold pieces placed on each of the paths to act as a baseline. Notice, that when using the multi-layered approaches an average of over 90% of the gold pieces appear on the provided path, compared to the maximum of 46.06% with the single-layer baseline. This shows that by providing a player path

Table 3: Multi-layer vs. Single-layer Gold on Path

Path	Single$_{std}$	Multi$_{std}$	Single$_{VLR}$	Multi$_{VLR}$
P0	38.17%	98.15%	40.22%	31.28%
P1	37.46%	100.00%	38.58%	100.00%
P2	31.29%	96.16%	33.45%	95.78%
P3	39.23%	99.67%	39.45%	99.71%
P4	39.67%	91.81%	37.99%	91.12%
P5	46.06%	98.49%	46.95%	99.58%
P6	35.71%	98.16%	35.68%	98.16%
P7	40.56%	95.68%	41.71%	98.14%
P8	21.81%	100.00%	23.11%	100.00%
P9	44.18%	97.43%	44.71%	97.49%
Avg.	37.42%	97.55%	38.19%	91.13%

layer, the multi-layer approach is able to sample a level with a solution similar to that provided path. A notable exception is with the VLR approach paired with P0. It is unclear why this particular path with multi-layer VLR approach performs so poorly, while it performs similar to other paths in the standard sampling approach, and in terms of the other metrics used.

Our results show that while the single-layer and multi-layer approaches perform similarly in terms of sampling time (sections resampled, levels finished) when using the VLR algorithm for generation, and the single layer approach even outperforms the multi-layer approach in terms of playability when using a standard sampling algorithm, the multi-layer approach provides the user with much more control over the type of levels created by the system, as evidenced by the placement of the gold pieces relative to the provided path. This suggests that the multi-layer approach can be of benefit when the user desires specific types of outputs from the system.

Conclusions and Future Work

In this paper we explored the application of our multi-layer level representation to the puzzle-platformer *Lode Runner*. We found that while the violation location resampling algorithm does much of the heavy lifting, in terms of ensuring usable levels are created, leveraging the multi-layer representation and providing a player path gives the user more control over the sampled levels than the with the single-layer approach. In the future, we would like to explore more diverse layers, such as difficulty layers, enemy behavior layers, and more informative section layers (perhaps via clustering). We would also like to continue exploring more complex domains, such as *Metroid* or *Mega Man*.

References

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

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

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

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

Katz, S. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing* 35(3):400–401.

Markov, A. 1971. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. In *Dynamic Probabilistic Systems: Vol. 1: Markov Models*. Wiley. 552–577.

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 *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Snodgrass, S., and Ontanon, S. 2015. A hierarchical MdMC approach to 2d video game map generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Snodgrass, S., and Ontañón, S. 2016a. Controllable procedural content generation via constrained multi-dimensional markov chain sampling. In *25th International Joint Conference on Artificial Intelligence*.

Snodgrass, S., and Ontañón, S. 2016b. Learning to generate video game maps using markov models. *IEEE Transactions on Computational Intelligence and AI in Games*.

Snodgrass, S., and Ontañón, S. 2017. Procedural level generation using multi-layer level representations with MdMCs. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*.

Summerville, A. J., and Mateas, M. 2015. Sampling Hyrule: Multi-technique probabilistic level generation for action role playing games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

Summerville, A.; Guzdial, M.; Mateas, M.; and Riedl, M. O. 2016a. Learning player tailored content from observation: Platformer level generation from video traces using lstms. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontanón, S. 2016b. The VGLC: The video game level corpus. In *Proceedings of the 7th Workshop on Procedural Content Generation*.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2017. Procedural content generation via machine learning (PCGML). *arXiv preprint arXiv:1702.00539*.

Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1(2):146–160.