Procedural Level Generation using Multi-layer Level Representations with MdMCs

Sam Snodgrass
College of Computing and Informatics
Drexel University
Philadelphia, Pennsylvania 19104
Email: sps74@drexel.edu

Santiago Ontañón
College of Computing and Informatics
Drexel University
Philadelphia, Pennsylvania 19104
Email: santi@cs.drexel.edu

***Abstract*—The use of statistical and machine learning approaches, such as Markov chains, for procedural content generation (PCG) has been growing in recent years in the field of Game AI. However, many of these level generation approaches account for only the structural properties of the levels. We developed multi-layered representations of levels, where each layer is designed to capture distinct gameplay information. Specifically, we experiment with representing levels using three layers: a *structural layer* that captures the object placements in the level, a *player path layer* that captures the path of an agent through the level, and a *height layer* that captures sections of the level as various height groups. We test our approach by generating levels for *Super Mario Bros.* with a multi-dimensional Markov chain approach. We compare the levels sampled with our multi-layered representation with those sampled using a single-layer approach that only considers the structural layer.**

I. 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) [1] 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 [2]–[5]. However, in most cases these approaches focus on learning structural information. Several of the aforementioned approaches use tile-based representations, which capture the structural components of the level. We build on this work by introducing additional layers of representation and generalizing a multi-dimensional Markov chain (MdMC) approach to allow for such level representations during training and generation.

Leveraging multiple layers of representation is not a new concept. Deep neural networks [6] often leverage multiple layers of representation to learn more accurate models of their domain. However, neural networks typically require a large amount of training data, and in PCGML there is often very little training data. Therefore, we need an approach that can leverage relatively small training sets. In the realm of PCGML, Snodgrass and Ontañón's [7] clustering-based hierarchical approach can be thought of as using multiple layers of representation, though they only focus on structural information in each layer. We are interested in capturing more than just structural level information. In this paper we propose an approach that represents training levels using multiple representation layers, each layer capturing a distinct aspect of the game level. We then propose extensions to the multi-dimensional Markov chain approach that allow it to be trained using the multi-layer training levels, and sample new levels.

The remainder of the paper is organized as follows. We start by formulating the specific problem we are trying to address in Section I-A. In Section II we give background on recent PCGML techniques, and other multi-layer machine learning approaches. In Section III, we discuss our multi-layer representation and multi-dimensional Markov chain approach. In Section IV we describe our experimental set-up and present our results. We close in Section V by drawing our conclusions and suggesting avenues of future work.

A. Problem Statement

In this paper we address the problem of representing more than just structural information while generating content. Specifically, we extend our level representation to model many different types of information, instead of only the commonly used structural information. We address this problem by developing additional layers of representation and extending the MdMC level generation approach to allow for any number of additional layers.

II. 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 [1]. This section discusses PCGML approaches and multi-layer machine learning approaches.

There have been several PCGML approaches for platformer level generation. Notably, Snodgrass and Ontañón used multi-dimensional Markov chains to sample levels for *Super Mario Bros*, *Lode Runner*, and *Kid Icarus* [5], where they account for current level height positions along with the level geometry by dividing the training data among several models. Summerville and Mateas [8] used long short-term memory neural networks, while Guzdial and Reidl [2] used Bayesian networks, both to generate *Super Mario Bros.* levels, where both approaches account for player movements; the LSTM through additional level annotations and the Bayesian model through implicit model parameters approximated by observing

player movements. Notice that the way all of these models include additional information is not easily extensible to other types of information that may be desired. We want to extend our model to allow for the learning of multiple layers of representation which can be adapted to many different types of information, including height and player path information.

The most notable approaches that use multiple layers of representation are neural networks [9]. Neural networks have been used to generate interesting images [10] and models [11]. However, neural networks typically require a huge amount of training data, and training data is often scarce for PCGML techniques. Therefore, we need an approach able to leverage small datasets.

III. Methods

In this section we first discuss our multi-layer representation, and then explain how we train and sample from an MdMC with the proposed level representations.

A. 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 $h \times w$ (height and width). Each layer has a separate set of tile types, T_i, where the meaning of the tiles varies by layer. For example, one layer may have tile types to represent the structures and objects in the level, while another layer may have tiles to represent paths through the level, and yet another layer could have tiles to represent meta-data about the level, such as heights.

Figure 1 shows a section of a *Super Mario Bros.* level 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 *height* layer (bottom-right) representing the height for each position in the level to help the model learn that patterns appearing at lower heights might be different from those at higher heights (Section IV describes these layers in more detail). 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 [2], [12] and occasionally player path information [13]. For example, in this paper in addition to a structural and a player path layer, we introduce a height layer which signifies different height sections of the level. However, these are only two possible additional layers. Others can include a difficulty layer, which could capture the difficulty of the level as you progress through it, or a dramatic tension layer which does the same for the dramatic tension.

B. Markov Chain-based Level Generation

We now explain how we train our model, and how we sample new levels using that trained model. We start with a brief introduction to multi-dimensional Markov chains, before discussing our training and sampling approaches.

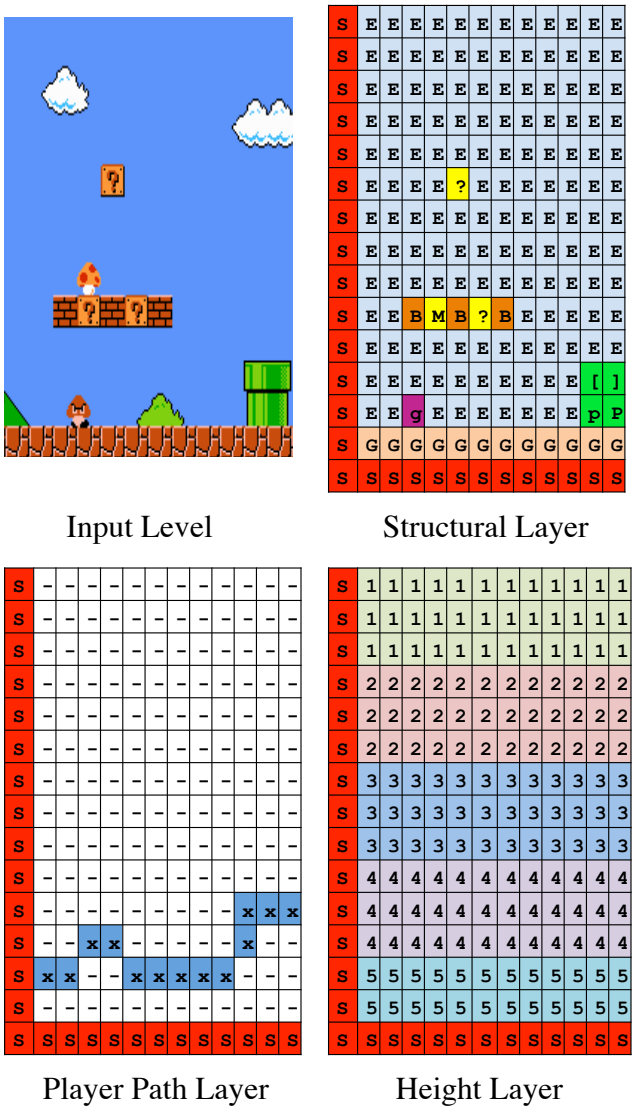

Fig. 1. This figure shows a section of a *Super Mario Bros.* level (top-left) represented using a structural layer (top-right), a player path layer (bottom-left), and a height layer (bottom-right). Color is added for clarity.

1) Markov Chains: Markov chains [14] 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 that the previous state was $S_{x-1} \in S$. The set of previous states that influence the CPD are referred to as the *network structure*.

Multi-dimensional Markov chains (MdMCs) are an extension of higher-order Markov chains [15] 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 tiles from other layers. For example, the CPD defining the MdMC in the right-hand side of Figure 2 (nsl_3) 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 correspond to the set of states in the various layers.

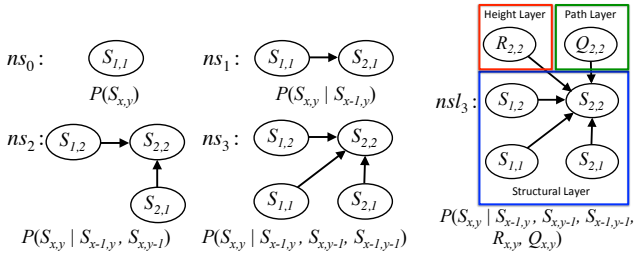

Fig. 2. This figure shows the network structures used by the single-layer MdMC (left), and a network structure used by the multi-layer MdMC. Note, that each nsl_i follows the same pattern as the corresponding ns_i, but the multi-layer network structures retain the dependencies from the other layers in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

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 [5].

2) Training: In this section we will first 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 two things: 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" for short. 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 example network structures that can be used to train a single-layer MdMC (left) and a multi-layer MdMC (right).

3) 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 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 look-ahead procedure. Instead of training a single MdMC model with a single network structure, we train a collection of them (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 [16]. More information on the look-ahead and fallback procedures as they apply to MdMCs can be found in [5].

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 need to be defined before sampling or computed during. We define the player path layer and height layers prior to sampling. Our goal is to guide the sampler towards creating levels that allow for the provided path. 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 height and player path layers. The fallback order for the multi-layer models is the same as in the single-layer approach.

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 [17] used in our experiments. This algorithm takes the desired dimensions of the output map and a set of constraints, C, and returns a map satisfying those constraints. Note that the constraints return 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 re-samples a new section, m, of the same

Algorithm 1 ViolationLocationResampling(w, h, C)

1: Map = MdMC([0, 0], $[w, h]$)
2: **while** $\left(\sum_{c \in C} c(Map).cost\right) > 0$ **do**
3: **for all** $c \in C$ **do**
4: **for all** $([x_1, y_1], [x_2, y_2]) \in c(Map).sections$ **do**
5: **for all** $c_i \in C$ **do**
6: $cost_{c_i} = c_i(Map[x_1, y_1][x_2, y_2]).cost$
7: **end for**
8: **repeat**
9: m = MdMC($[x_1, y_1], [x_2, y_2]$)
10: **for all** $c_i \in C \setminus c$ **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: $Map[x_1, y_1][x_2, y_2] = m$
17: **end for**
18: **end for**
19: **end while**
20: **return** Map

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).

IV. Experimental Evaluation

We test our approach by sampling levels for the classic video game, *Super Mario Bros.* Our goal is to determine whether our multi-layer approach is able to more accurately model and recreate interesting situations from the training data than the single-layer MdMC approach.

The remainder of this section describes the chosen domain and the experimental set-up, and reports our results.

A. Domain

Super Mario Bros. is a platforming game with linear levels, meaning the player progresses through the level in one direction (e.g., left to right) [3]. In our experiments we use a combined 25 levels from *Super Mario Bros.* and *Super Mario Bros.: The Lost Levels*. These levels could be found in the video-game level corpus[1] [18].

B. Level Representation

In this section we describe the layers we use to represent our training levels and how we create those layers.

1) Structural Layer: The structural layer is the main layer of the representation, and captures the placement of objects and enemies through the level. This layer is represented by an array, where each cell takes the value from a set of 34 tile types corresponding to the elements in the level (e.g., pipes, enemies, blocks, springs, etc.). It is important to note that in these experiments we use a more expressive set of tiles than has been used in previous work. Specifically, previous MdMC experiments [5] used a set of 7 tile types and LSTM experiments [8] used a set of 13, both of which abstracted away many details from the levels, such as different enemy and block types, as well as removing springboards and moving platforms. In this paper we expand the tile set to include indicators for moving platforms, individual tile types for the various enemy types, springboards, and tile types indicating what each block contains. Figure 1 (top-right) shows a section of a level represented using the structural layer.

2) Height Layer: The height layer is one of the additional layers, and captures the various height sections of the level. That is, it essentially splits the level into multiple sections by grouping rows of the level together to allow for more focused training to occur within each section. This works similarly to the *row split* parameter used by Snodgrass and Ontañón in their previous work [12]. For our experiments we use a set of 6 tile types, where 4 tile types are each used for 3 consecutive rows, one is used for the final two rows, and the final tile type is a sentinel tile that signifies the boundaries of the layer. Figure 1 (bottom-right) shows a section of a level represented with the height layer.

3) Player Path Layer: The player path layer is the final layer. This layer captures a possible path through a level, and is represented using 3 tile types: "x" represents a position in the level that the path passes through, "-" denotes positions not on the path, and "S" is a sentinel tile signifying the boundaries of the layer. During training, we use Summerville et al.'s A^* agent [4] to create the player path layer for the training levels. During sampling, we experimented with providing path layers of the A^* agent traversing a training level as well as hand authored path layers; these paths are discussed more in the following section. Figure 1 (bottom-left) shows a section of a level representing with the player path layer.

C. Experimental Set-up

We tested our multi-layer approach by first training a single-layer and a multi-layer MdMC (as described in Section III-A) on 25 training levels. The single-layer and the multi-layer MdMCs require several parameters to be set before they can be trained and used to sample new levels.

- **Single-layer MdMC**: This requires setting a look-ahead, a row-split, and a set of network structures to be used during training and for the fallback during sampling. In our experiments we use a look-ahead of 3 and a row-split of 5. For the network structures, we use ns_3 as the main network structure. During sampling, ns_3 will fallback to ns_2 which falls back to ns_1, which finally falls back to ns_0, the raw distribution of tiles in the structural layer. These network structures can be seen in Figure 2 (left).
- **Multi-layer MdMC**: This requires a look-ahead value, a set of network structures to be used during training and as fallbacks during sampling, a player path layer to be used during sampling, and a height layer to be used during sampling. In our experiments, we use a look-ahead of 3.

[1] https://github.com/TheVGLC/TheVGLC

We use nsl_3 as our main network structure (as seen in Figure 2, right). For the fallback network structures, we use nsl_2, which falls back to nsl_1, which falls back to nsl_0. Recall, the nsl_i network structures are defined the same way as the ns_i network structures, with the addition of dependencies on the height and player path layer. During sampling, we provide the height layer shown in Figure 1 (bottom-right), and various player path layers. A description of the player path layers we use during sampling can be found later in this section.

We use the trained model paired with the violation location resampling (VLR) algorithm to enforce two constraints:

- **Playability()**: To satisfy this constraint, a path must exist from the beginning to the end of the level. We test this with a version of Summerville et al.'s A^* agent [4] augmented to account for springboards, which change the movements possible for the agent.
- **Wellformedness()**: To satisfy this constraint, each pipe, cannon, and springboard must be well formed in the level. That is, pipes must have a width of 2, with pipe tops placed accordingly; cannons must have bullet-bill shooters on top; and springboards must have both the bottom and top portion of the springboard. Additionally, all of these must be placed correctly on solid tiles.

For our experiments, we sampled 1000 levels with the single-layer MdMC approach, and 1000 levels with the multi-layered MdMC approach using 4 different player path layers (i.e. 250 levels sampled for each player path layer). For the player path layers, we used 2 paths given by the A^* agent: the first for *Super Mario Bros.* level 1-1, the first level; and the second for *Super Mario Bros.* level 2-1, which includes a springboard. Additionally, we used 2 hand-crafted player path layers: the first, is a simple plateau-shaped path, requiring a single jump and fall; the second, is a more complex path including many jumps of various height and distances[2].

We are interested in evaluating how well our multi-layer approach is able to model interesting interactions from the more expressive tile representation we are using in our structural layer. To evaluate this, we use "spring boards" (an infrequent tile type that allows the main character to perform very high jumps), and calculate the ratio of springs in the sampled maps and the number of springs in the sampled maps that are required to complete the level (i.e. we want to see whether those spring boards were included because they were necessary for the specified player path, or if they were inserted by chance). We are also interested in evaluating how well our multi-layer model, given a player-path layer, is able to sample a level allowing for that path. Therefore, we compute the discrete Fréchet [19] distance between the provided path and the actual path taken through the sampled level using an A^* agent. Finally, we would like to know how well our approach models the training data. To accomplish this we show a t-SNE [20] visualization of the training levels and sampled levels, and also compare the linearity and leniency [21] of the sampled levels against the training levels. These metrics are described more fully below:

[2]This data is available at sites.google.com/site/sampsnodgrass/

TABLE I
LEVEL EVALUATION

	Linearity	Leniency	Fréchet	Springs
Training	2.309	0.151	–	0.440/27.3%
Single	2.213	0.160	8.456 (P1) 12.54 (P2) 8.696 (P3) 30.58 (P4)	0.350/0.9%
Multi P1	2.168	0.161	6.222	0.144/5.6%
Multi P2	2.180	0.168	8.144	0.272/1.5%
Multi P3	2.461	0.150	6.219	0.212/5.7%
Multi P4	2.251	0.166	6.794	0.220/5.5%

- **Linearity**: This measures how well the platforms in the level can be approximated with a best fit line. It returns the sum of distances of each solid tile type from the best-fit line, normalized by the level length.
- **Leniency**: This approximates the difficulty of the level by summing the gaps (weighted by length) and enemies (weighted by 0.5), and normalizing by the level length.
- **Fréchet** [19]: This measures the distance between two paths. Intuitively it can be thought of as the minimum length of a rope needed to connect two people walking on two separate paths over the course of the paths.

In our experiments, we want our sampled levels' linearity and leniency to closely match the training levels' linearity and leniency. Alternatively, we want the Fréchet distance between the provided path and the actual path to be minimized.

D. Results

Table I shows the results of our experiments. The linearity and leniency columns show the averages of the values computed over all the levels in each set of levels. The Fréchet column shows the average Fréchet distance between the path provided in the player path layer to the multi-layer MdMC and the path taken by the A^* agent through the sampled levels over all the sampled levels. For the single-layer model we compute the Fréchet distance between the four paths used for the multi-layer models and the path found for each of the sampled single-layer levels. The final column shows the average number of springs per level and the percentage of those springs that are required to complete the level. The rows correspond to the training levels, the levels sampled using the single-layer MdMC, and the levels sampled using the multi-layer MdMC model paired with each of the four provided player path layers.

First, notice that the single-layer MdMC and the multi-layer MdMC approaches were able to achieve linearity and leniency values close to the training levels. This is further reflected in Figure 3, which shows the expressive ranges of the sampled levels with respect to linearity (x-axis) and leniency (y-axis), and shows that all the models achieve a similar expressive range. This indicates that both of the models are able to accurately model the structural elements of the training levels. However, if we look to the Springs column in Table I, we see

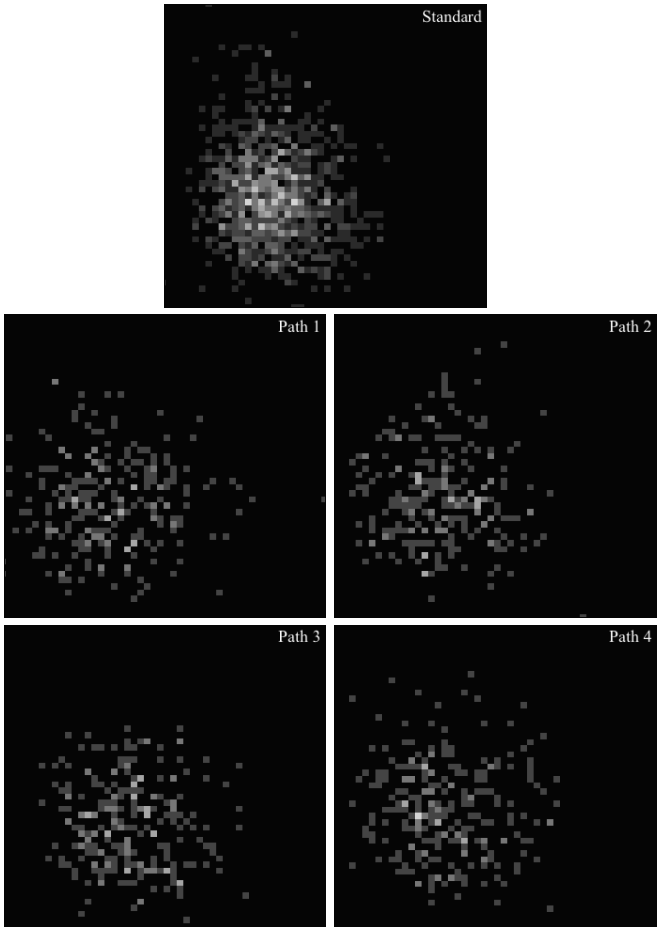

Fig. 3. This figure shows the expressive range of the different models. The y-axis is the leniency of the sampled levels and the x-axis is the linearity of the sampled levels; all values are normalized to between 0 and 1. *Standard* refers to the single-layer model, while *Path 1* - *Path 4* refer to the multi-layer models using the different paths.

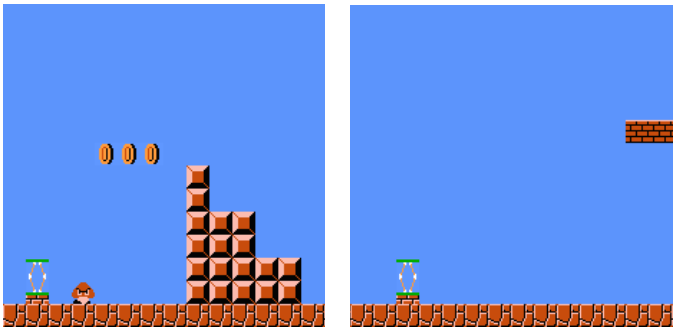

Fig. 4. This figure shows a section of level sampled using the multi-layer P1 MdMC model where a spring is need to complete the level (left), and a section of a level sampled using the single-layer MdMC model where a spring is placed, but not needed to complete the level (right).

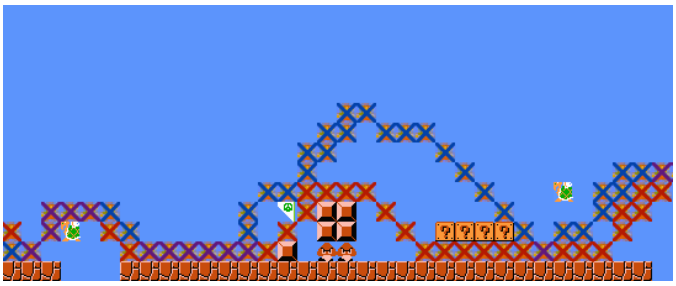

Fig. 5. This figure shows a section of a level sampled with the multi-layer P2 MdMC that achieved a low Fréchet distance (4.123), along with the path provided to the sampler for that level (blue), the actual path taken by the A^* agent through the completed level (red), and the overlap of the two paths (purple). Notice, there is a fair amount of overlap in the paths, and even when not overlapping, the paths do not stray far from each other.

that the single-layer MdMC has a very difficult time placing springboards in meaningful places. Alternatively, the multi-layer MdMC models place springboards much more reliably (typically around 5% instead of 0.9%). This shows that the multi-layer model is able to capture the nuances about how the springboards function better than the single-layer model. Figure 4 shows a section of a level sampled with the standard approach that placed a springboard where it is not needed (right) and a level sampled with the multi-layer approach that placed a necessary springboard (left).

Next, the Fréchet distances for all of the multi-layer models are lower than when comparing the provided paths against the A* paths taken through the single-layer models. This shows that our multi-layer approach is able to reliably generate levels that allow for the path provided to the model. In particular, notice that the Fréchet distance between single-layer model levels' A* paths and P4 is much larger (30.58) than the other paths. This is likely due to the complexity of P4, which requires many large and small jumps. However, the multi-layer model (Multi P4) is still able to achieve a low Fréchet distance (6.794) for this complex path. Figure 5 illustrates this result with a section of a level sampled using the multi-layer MdMC approach annotated with the provided and actual paths through the level. We see there is a fair amount of overlap in the paths, and when not overlapping the paths remain near each other.

Figure 6 shows a two-dimensional projection of the evaluated levels. Maps were projected based on a measure of distance between them using the t-SNE visualization algorithm [20]. To determine the distance between two maps, we represented them as a histogram of high-level tiles, and computed the Euclidean distance between these histograms. High-level tiles were found by clustering 4×4 tile sections using k-medoids ($k = 30$) with all the training levels, 20 levels sampled by the single-layer approach, and 5 levels sampled by each of the multi-layer models. For k-medoids, we used a distance metric that compares the shape of the structures in the two sections and returns the difference between those shapes scaled by the overlap of the sections (more information about using k-medoids for clustering levels sections can be found in [7]). We have highlighted training levels 1 and 2, because Multi-layer P1 and P2 (Multi P1 and Multi P2 in the figure), respectively, used the paths from these training levels in their player path layers. Notice that training level 2 is occluded by a level sampled by multi-layer P2, and further that those training levels are surrounded by the levels sampled by P1

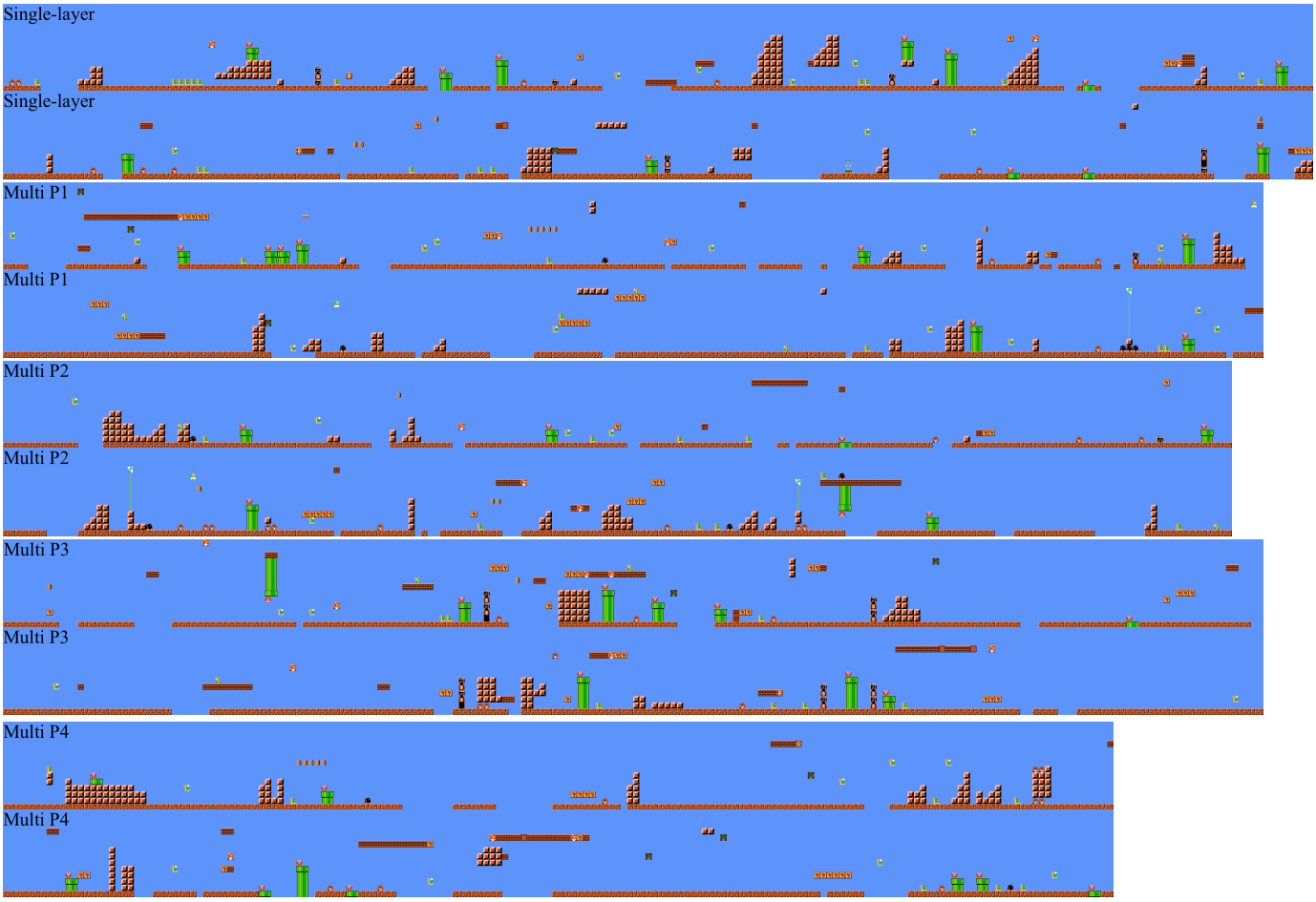

Fig. 7. This figure shows two levels sampled with each of the models: single-layer MdMC (top two), and the multi-layer MdMC using each of the four provided paths (from top to bottom).

and P2. This shows that the multi-layer models are able to capture information about the training levels more accurately than the single-layer approach. Furthermore, there are only a few single-layer sampled levels near training levels 1 and 2, which shows that our multi-layer MdMC more accurately models these levels than the single-layer approach. Finally, the levels sampled by multi-layer P4 (Multi P4 in the figure) are separated from the rest of the sampled levels, but still near several of the training levels indicating that if a new unseen path is provided to the multi-layer MdMC it is able to produce reasonable, but different levels.

Lastly, Figure 7 shows 2 randomly selected sampled levels for the single-layer MdMC and each of the multi-layer MdMC models, with the A^* agent's path annotated. Notice, the levels sampled using the multi-layer MdMC with the plateau-shaped path (Multi P3 in the figure) have very few structures at the beginning and end of the levels, but many in the middle. This allows the agent to follow a path close to the one provided. Similarly, the other levels sampled using the multi-layer MdMC models place structures near where jumps are designated in the provided paths.

Our most important result is that our multi-layer MdMC model is able to sample levels according to a provided player path, while still producing levels similar to the training levels. This shows that our multi-layer MdMC approach is a viable way to more accurately and deeply model video-game levels than similar single-layer models, opening the door for other representation layers to be developed and applied.

V. Conclusion

In this paper we presented a multi-layered video-game level representation and introduced an extension to the multi-dimensional Markov chain PCG approach that allows it to train using levels with multiple layers of representation. Namely, in addition to the standard structural layer, we experimented with a player path layer and a height layer. We tested our approach by sampling levels for *Super Mario Bros.* We found that the multiple layers of representation allow our model to capture nuances, such as springboard placement, more easily and accurately than a similar single-layer approach. Further, we found that we are able to guide the sampler towards levels that allow for specific paths through the level by providing the desired path to the model at the time of generation.

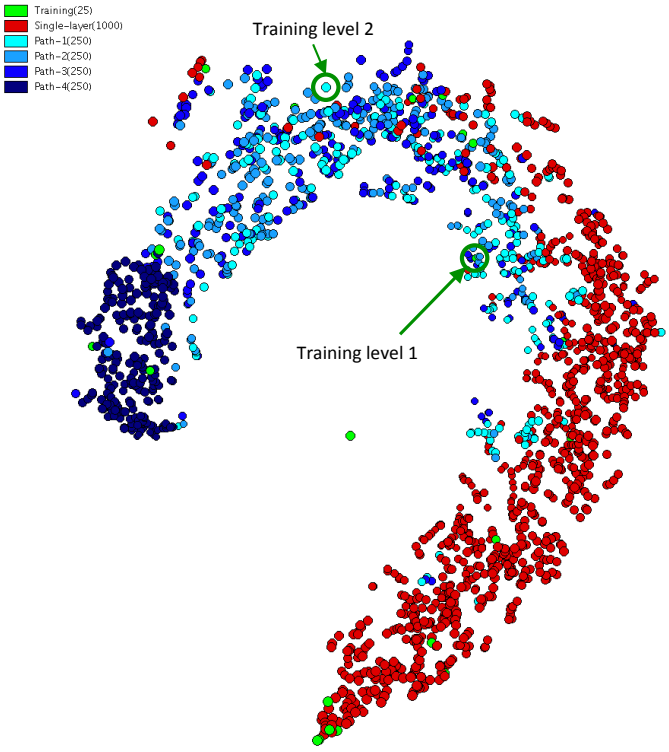

Fig. 6. This figure shows the two-dimensional projection of the training levels and sampled levels. We highlight the first two training levels because the paths through these levels were used as the provided player path layers for the first two multi-layer models. Notice, the two selected training levels are occluded by generated levels.

In the future, we would like to explore more layers of representation capturing interesting meta-data about the levels. In particular, we are interested in exploring layers representing level progression, such as difficulty or drama curves. Additionally, We would also like to explore the use of this multi-layered approach in domains where the single-layer approach has previously had difficulty, such as *Lode Runner*, where path information and item collection is important.

REFERENCES

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

[2] M. Guzdial and M. Riedl, "Game level generation from gameplay videos," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

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

[4] A. J. Summerville, S. Philip, and M. Mateas, "MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

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

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

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

[9] S. Haykin and N. Network, "A comprehensive foundation," *Neural Networks*, vol. 2, no. 2004, p. 41, 2004.

[10] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," *Google Research Blog. Retrieved June*, vol. 20, p. 14, 2015.

[11] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1538–1546.

[12] S. Snodgrass and S. Ontañón, "Experiments in map generation using Markov chains," in *Proceedings of the 9th International Conference on Foundations of Digital Games*, vol. 14, 2014.

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

[14] A. Markov, "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, 1971, pp. 552–577.

[15] W.-K. Ching, X. Huang, M. K. Ng, and T.-K. Siu, "Higher-order Markov chains," in *Markov Chains*. Springer, 2013, pp. 141–176.

[16] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE transactions on acoustics, speech, and signal processing*, vol. 35, no. 3, pp. 400–401, 1987.

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

[18] A. J. Summerville, S. Snodgrass, M. Mateas, and S. Ontanón, "The vglc: The video game level corpus," in *Proceedings of the 7th Workshop on Procedural Content Generation*, 2016.

[19] T. Eiter and H. Mannila, "Computing discrete fréchet distance," Citeseer, Tech. Rep., 1994.

[20] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[21] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 2010, p. 4.