Player Movement Models for Platformer Game Level Generation

Sam Snodgrass, Santiago Ontañón
Department of Computer Science, Drexel University
Philadelphia, PA USA
sps74@drexel.edu, 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, there has been little work in learning to generate content, specifically levels, accounting for player movement within those levels. We are interested in extracting player models automatically from play traces and using those learned models, paired with a machine learning-based generator to create levels that allow the same types of movements observed in the play traces. We test our approach by generating levels for *Super Mario Bros.* We compare our results against the original levels, a previous constrained sampling approach, and a previous approach that learned a combined player and level model.

1 Introduction

Procedural content generation (PCG) studies the algorithmic creation of content (e.g., maps, textures, music, etc.), often for video games. Recently there has been increased interest in the use of machine learning-based approaches to create video game content [Guzdial and Riedl, 2016; Dahlskog *et al.*, 2014; Summerville *et al.*, 2015] also called PCG via machine learning or PCGML [Summerville *et al.*, 2017]. However, most of these approaches are based on modeling level geometry, without taking into account how the player moves through levels, which is an important part of level design. The problem we address in this paper is how to leverage these machine learning approaches to generate maps that account for player movement.

Player models have been used to guide content generators towards player-specific content, called *experience-driven PCG* [Yannakakis and Togelius, 2011]. However, most experience-driven approaches rely on player models paired with search-based content generation algorithms [Togelius *et al.*, 2007; Yannakakis and Togelius, 2011]. We are interested in leveraging machine learning both in the player model extraction as well as for training the content generator. Summerville *et al.*'s recent approach [Summerville *et al.*, 2016] is a notable exception in which they developed an approach that annotated training maps with human paths extracted from gameplay videos, and generated maps with a model trained on those annotated maps, learning an implicit player movement model within their generator. In this paper we propose an approach that disentangles the player movement model from the map model, allowing the movement model to be used to guide the map model during generation, as well as to evaluate paths through maps.

The remainder of the paper is organized as follows. We start by formulating the specific problem we are trying to address in Section 1.1. Then, in Section 2 we give background on recent PCGML techniques for level generation and player modeling approaches within the context of PCG. Next, in Section 3 we discuss our Markov chain-based map generation approach before moving onto our player movement modeling approach in Section 4. Afterwards, in Section 5 we describe our experimental setup and discuss our results. We close in Section 6 by drawing our conclusions and suggesting avenues of future work.

1.1 Problem Statement

In this paper we address the problem of accounting for player movement while generating content. Specifically, we are interested in the problem of generating levels that allow for the types of movements observed in example play traces. We address this problem by automatically extracting player movement models from gameplay traces, and using those models to guide a statistical level generator towards levels with high-likelihood paths according to the learned model.

2 Background

Procedural content generation (PCG) is the algorithmic creation of content, typically for video games [Shaker *et al.*, 2015]. This section discusses machine learning-based approaches and experience-driven approaches.

We are interested in generators that learn statistical properties from training data (i.e., existing maps), and use them to sample new content, that is procedural content generation via machine learning (PCGML) [Summerville *et al.*, 2017]. For example, Dahlskog et al. [2014] proposed sampling new maps using n-grams trained on input maps. Related approaches include generating maps using a model trained on gameplay footage [Guzdial and Riedl, 2016], and treating maps as strings and training a recurrent neural network in order to sample new maps [Summerville and Mateas, 2016].

There has also been work on combining multiple machine learning techniques to sample action RPG levels at multiple levels of detail [Summerville and Mateas, 2015] as well as learning ways of combining generators to achieve desired content [Shaker and Abou-Zleikha, 2014].

We are also interested in experience-driven PCG approaches [Yannakakis and Togelius, 2011], or approaches that use player models to guide generators. Yannakakis et. al [2011] discuss player models for *Super Mario Bros.* and how those model are used to guide an evolutionary map generator. This can also be seen in *Galactic Arms Race* where the use of the various guns affects the types of guns that get generated [Hastings *et al.*, 2009], and in race track generation [Togelius *et al.*, 2007]. These approaches use player models paired with evolutionary algorithms, but we want to leverage machine learning to build player models and level models, then use both models to generate new content.

Recently, Summerville *et al.* [2016] developed an approach that incorporates player behavior into the training maps in the form of a path, then trains an LSTM neural network on those annotated maps. This allows them to capture an implicit player model within their map model, which they then use to generate new maps. We will refer to this approach as the *SGMR* approach for the remainder of this paper (after the authors' names). The work we present in this paper differs from theirs by disentangling the player movement model from the map model. This allows us to both evaluate potential player paths and more explicitly guide our generator, using the learned player movement model.

3 Markov Chain-based Map Generation

In this section we give a brief introduction to multi-dimensional Markov chains (MdMCs). We then discuss how they are used to model and sample video game maps.

3.1 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) $P(S_t|S_{t-1})$, representing the probability of transitioning to a state $S_t \in S$ given that the previous state was $S_{t-1} \in S$. The set of previous states that influence the CPD are referred to as the *network structure* of the model.

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. For example, the CPD defining the MdMC in Figure 1 (ns_3) can be written as $P(S_{t,r}|S_{t-1,r}, S_{t,r-1}, S_{t-1,r-1})$. By redefining what a previous state can be in this way, the model is able to more easily capture relations from two-dimensional training data, as shown in our previous work [Snodgrass and Ontañón, 2016b].

3.2 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 T, the set of tile types which correspond to the states of the MdMC and the objects in the map. We add sentinel tiles to signify the boundaries.

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
```

3.3 Training

Training an MdMC requires a network structure and training maps. Figure 1 shows example network structures that can be used to train an MdMC. Training happens in two steps: *Absolute Counts* and *Probability Estimation*. First, given the network structure, we determine the tile configuration (i.e., positions and types of the previous tiles) for a position in the map. We then count the number of times each tile follows each tile configuration. Next, the conditional probability distribution that defines the MdMC is estimated from these counts.

3.4 Sampling

Given a desired map size, $h \times w$, a map is sampled one tile at time, starting, for example, in the bottom left corner, and completing an entire row before moving onto the next row. For each tile, the MdMC is used to sample a tile based on the tile configuration and the trained probability distribution.

While sampling, we use a *look-ahead* and *fallback* procedure that generates a number of tiles ahead trying to avoid *unseen states* (i.e., combinations of tiles that were not observed during training, and that we, thus, do not have a probability estimation for). More information on this sampling approach can be found in [Snodgrass and Ontañón, 2016b].

For the approach presented in this paper, we train a player movement model which can determine the likelihood of a given path through a map. Our goal is to sample maps that allow for high-likelihood paths. In order to accomplish this, we employ an extension [Snodgrass and Ontañón, 2016a] to the standard sampling approach that is able to sample maps satisfying provided constraints; in our case, one constraint is a high-likelihood path through the sampled map. Below we discuss the constrained sampling algorithm in more detail.

Algorithm 1 shows the *Violation Location Resampling* or *VLR* algorithm used in our experiments. This algorithm takes the desired dimensions of the output map and a set of con-

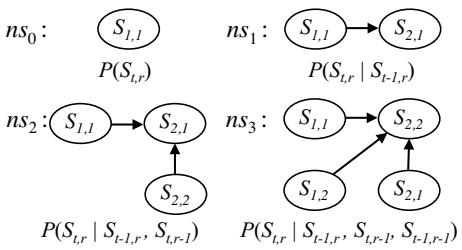

Figure 1: The network structures used in our experiments; ns_3 is used as the main network structure, and each preceding ns_i is used as a fallback network structure.

straints, 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 map, Map, using the standard sampling approach (line 1). Next, if any constraints return a nonzero cost for the map (line 2), then for each constraint, $c \in C$ (line 3), the algorithm iterates over the sections of the map 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), and checks if the cost of m is greater than the previous cost for any other constraint (lines 10-14). If so, m is resampled, and cost checking is repeated. If the cost with regards to the other constraints is not raised, then m is only accepted if the cost with regards to the current constraint, c, is lowered (line 15). This process of finding violated sections and improving their costs is repeated until the total cost of Map is 0 (line 2).

4 Player Movement Model

Our player movement model captures the probability of the player-character performing a specific action. Formally, we define a player movement model as a finite set of actions $A = \{a_1, a_2, ..., a_n\}$, and a conditional probability distribution (CPD) conditioned on either the previous action, the current surroundings of the player-character, or both. We define an *action* as the operation performed by the player-character in a given time-step (e.g., movement, interacting with an objects, etc.). We define a *surrounding* as the area in the map around the player-character. Specifically, when using tile maps, a surrounding, s, is an $x \times y$ window of tiles with the player-character at the center. Given that we use a finite tile set to define our maps and our maps are of finite size, there are a finite set of surroundings possible, denoted S. Therefore, we can define the CPD of the player movement model as $P(A_t|A_{t-1})$, when conditioned on the previous action; $P(A_t|S_t)$, when conditioned on the current surroundings of the player-character; or $P(A_t|A_{t-1}, S_t)$, when conditioned on both, where t indicates the current time-step

To build the above conditional probability distributions, we extract sequences of actions and accompanying surroundings from gameplay videos. Below we explain how we extract the actions and surroundings from a gameplay video, and then discuss how we train our player movement models.

E	E	E	E	E
E	E	?	E	E
E	E	E	E	E
E	E	E	E	E
E	E	E	E	E
B	?	B	?	B
E	X	E	E	E
E	E	E	E	E
E	E	E	E	E
G	G	G	G	G

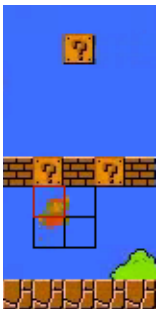

Figure 2: A section of a frame taken from a gameplay video of *Super Mario Bros.* (right) and our tile representation (left). Notice that though the player-character does not completely occupy one tile position, its tile representation is placed into the most fully occupied position (denoted by "X").

4.1 Play Trace Extraction

As discussed in Section 2, experience-driven procedural content generation leverages player models in order to guide generators towards desirable content. We are interested in automatically learning one such player movement model from gameplay videos. We will start with explaining how we extract the sequence of actions from the videos and then how we extract the surroundings.

We start, similarly to the *SGMR* approach, by converting the gameplay video into individual frames. We then represent each frame as an $h \times w$ tile map, with a special tile type representing the player-character's position in the map. Note that while the position of the player-character in the tile map must be discrete, the player-character's movement in the video may be continuous, resulting in the player-character not falling exactly into one position in the tile representation. To remedy this, we determine which tile position contains the most of the player-character, and place the player-character's tile representation in that position in the tile map. An illustration of this process can be seen in Figure 2. This process is also used to place other moving elements, such as enemies.

Once we convert all the frames into their tile representations, we can extract the action and surrounding sequences. To extract the action sequence, we compare sequential pairs of frames. First, we align them according to their level geometry, and then determine the difference between the positions of the player-character in each frame. This gives us the action taken between the two frames (e.g., moving right, jumping up, standing still). We repeat this process for each sequential pair of frames to get the sequence of performed actions.

To extract the sequence of surroundings, we examine each frame's tile representation individually. We locate the position of the player-character's tile and extract an $x \times y$ tile window centered at that position. If the window extends beyond the edges of the tile frame, we fill those positions with sentinel tiles. Repeating this for each frame gives us a sequence of surroundings. Because there are many possible surroundings, we perform k-medoids clustering [Park and Jun, 2009] with $k = 20$ on the observed set of surroundings to find exemplar surroundings to be used during training.

4.2 Model Training

Given action and surrounding sequences we can now train our player movement models. We propose three models:

- **Actions Only**: This model learns the probability of performing an action given the previous action. That is, it learns $P(a_t|a_{t-1})$, where $a_i \in A$, and A is the finite set of all observed actions.
- **Surroundings Only**: This model learns the probability of performing an action given the surroundings of the player (i.e., the map geometry). That is, it learns $P(a_t|s_t)$, where $a_i \in A$, as above, and $s_i \in S$, and S is the finite set of exemplar surroundings.
- **Actions and Surroundings**: This model learns the probability of performing an action given the previous action and the current surroundings. That is, it learns $P(a_t|a_{t-1}, s_t)$, where $a_i \in A$, $s_i \in S$, as above.

To train these models, we estimate the conditional probability distribution for the model according to the frequency of occurrences in the gameplay video via the extracted action and surrounding sequences. That is, we count how many times each action follows each condition (either action, surrounding, or pair of action and surrounding), and then set the probability of each action occurring following each condition according to the observed counts. The code used to train our models and perform our experiments are available online[1].

5 Experimental Evaluation

We test our approach by sampling maps for the classic video game, *Super Mario Bros.* The remainder of this section describes the chosen domain, elaborates on the experimental set-up, and reports our obtained results.

5.1 Domain

Super Mario Bros. is a platforming game with linear maps (as defined by [Dahlskog *et al.*, 2014]). That is, the player traverses the maps from left to right while avoiding enemies and pits. We extracted play traces for 4 maps using the method outlined in Section 4, for a total of $2,685$ frames. We extracted the play traces from a gameplay video posted online[2] of a single human player playing through the game.

5.2 Experimental Set-up

We tested our approach by training an MdMC on the 4 maps for which we had play traces. The MdMC approach allows for the configuration of several parameters to improve performance depending on the domain [Snodgrass and Ontañón, 2016b]. For our experiments, we set the parameters as follows: $rowsplits = 14$, the height of the maps; $lookahead = 3$; and using the network structure n_3, seen in Figure 1, and falling back to n_2, n_1, and n_0 as needed. We use the trained MdMCs paired with the violation location resampling (*VLR*) algorithm in order to enforce 2 constraints:

[1] bitbucket.org/Sam_Snodgrass/ijcai_2017

[2] youtube.com/watch?v=bNNwNPUzCMo

Table 1: Likelihood of Paths through Maps

	L_A	L_S	L_{AS}
TD_{Player}	0.22592	0.14577	0.23890
TD_{Agent}	0.12872	0.14512	0.11846
$VLR_{Playability}$	0.12687	0.15300	0.11894
$VLR_{Likelihood}$	**0.16079**	**0.19128**	**0.15204**
$SGMR_A$	0.13388	0.15630	0.12134
$SGMR_B$	0.13133	0.14751	0.11740
$SGMR_C$	0.13265	0.15402	0.12043
$SGMR_D$	0.13352	0.15893	0.11801
$SGMR_{Avg}$	0.13284	0.15419	0.11929
$SGMR_{All}$	0.13750	0.15817	0.12261

- **Playability**(): To satisfy this constraint, a path must exist from the beginning to the end of the map. We test this with Summerville *et al.*'s A* agent [Summerville *et al.*, 2015]. Unplayable sections are returned for resampling.
- **Likelihood**(min): To satisfy this constraint, the path through the level found by the A* agent must have a likelihood above min, as evaluated by a specified player movement model defined in Section 4. The lowest likelihood sections are returned for resampling.

In order to keep our *Surroundings Only* and *Actions and Surroundings* player movement models reasonable, we performed k-medoids clustering (with $k = 20$) using the 5×5 windows surrounding the player in each frame as the objects to cluster. We found in preliminary experiments that 20 clusters were enough to capture most of the various structures found in the training maps, and that 5×5 windows captured enough of immediate surrounding information.

For our experiments, we use the *Actions and Surroundings* player model to evaluate the likelihood of the agent's path during sampling. We chose the minimum value of 0.15 based on preliminary experimental results. We then sampled 100 maps using the *VLR* algorithm paired with only the playability constraint and 100 maps with the *VLR* algorithm paired with both the playability constraint and the likelihood constraint. We compare the likelihood of the A* agent's path through our sampled maps against the observed player's path through the training maps, the A* agent's path through the training maps, as well as the A* agent's path through 100 levels sampled by the *SGMR* approach using various play traces.

5.3 Results

Table 1 shows the results of our experiments. L_A refers to the average likelihood of the A* agent's path through the maps evaluated by the *Actions Only* player movement model, L_S refers to the same using the *Surroundings Only* model, and L_{AS} refers to the same using the *Actions and Surroundings* model. TD_{agent} and TD_{player} refer to the training maps' paths for both agent and player, respectively; $VLR_{Playability}$ and $VLR_{Likelihood}$ refer to the our sampled maps' paths, using only a playability constraint and with both playability and likelihood constraints, respectively; and *SGMR* refers to the paths of maps sampled by the *SGMR* approach using each of their play traces for training (A-D), the combined values for the maps sampled using each of those videos (Avg.), and the

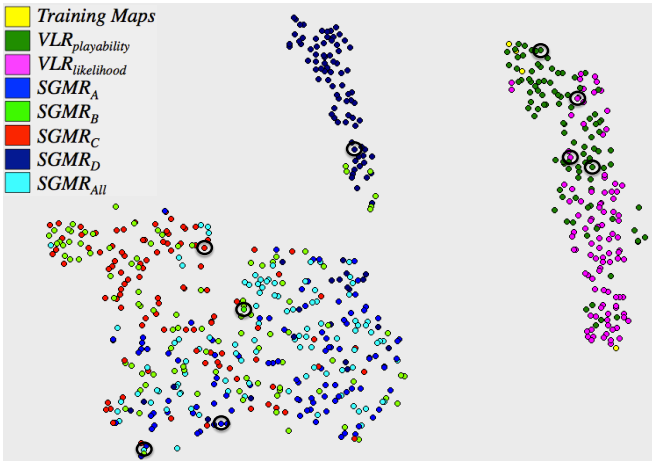

Figure 3: 2-D projection of the 4 training maps, the maps sampled with our approach, and maps sampled with the *SGMR* approach. Circled maps appear in Figure 5.

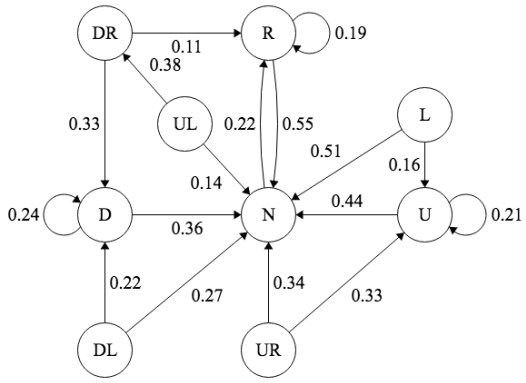

Figure 4: This figure shows a probabilistic finite state machine where each node is an action. This represents our *Action Only* model. "R" is moving right, "L" is moving left, "U" is moving up, "D" is moving down, "N" is no movement, and the combinations are diagonal movements. For clarity, we only include each node's two most probable transitions.

maps sampled after training on all of the videos (All). Note that we randomly selected maps from sets of maps generated by the *SGMR* approach with each of play traces until we had 100 maps that were able to be completed by the A* agent for each of the configurations. We used the A* agent's paths in these maps instead of the paths generated by their method in order to ensure uniformity throughout our evaluation.

First, it is important to note that, in general, the paths obtained from the A* agent are much less likely than the player's path (when comparing the paths in the training maps). This is to be expected, as the evaluation models were trained on the observed player's path, and an A* agent is unlikely to behave very similarly to a human player. However, when only accounting for the current surroundings of the player, the likelihoods of the player and agent paths are similar (again for the training maps' paths). This is because when considering only the surroundings, there are often situations with one obvious solution. For example, if approaching a pit with no other obstacles, it is likely for the player or agent to jump; similarly, if on a flat surface with no other obstacles, it is likely for the player or agent to move forward. This movement model is too simplified though, as it only captures what the player might do given the current surroundings without accounting for what may have happened immediately previous. For example, if the player-character is in the air, then only accounting for the surroundings it is difficult to predict if she is moving up (jumping) or down (falling). With knowledge of the previous action, this prediction is much easier, as we can see in which direction she was moving previously.

Second, notice that the likelihood of the agent's path is fairly uniform across all of the maps, except for the maps sampled by the *VLR* algorithm enforcing a playability and likelihood constraint. This is to be expected, as this approach will only sample maps that have a path likelihood above a threshold. However, while our method guarantees a certain level of path likelihood, we are also interested in investigating how the structures of the various maps relate.

Figure 3 shows the sampled maps projected in a two-dimensional space based on a measure of distance between them using the t-SNE visualization algorithm [Maaten and Hinton, 2008]. 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 the four training maps and one map from each of the sampled sets. From the projection, we can see that our sampled maps and the other sets of sampled maps are quite distinct from one another, and that our sampled maps lie closer to the training maps, while the other maps are separated from them. Further, the maps we sampled using the likelihood constraint lie closer to one particular training map, while the maps sampled using only the playability constraint lie closer to the other three training maps. This is likely due to the structure of the training maps. The one training map is flat with very few other structures implying that to create maps with high-likelihood paths, our model generated maps with stretches of flat space (as moving forward on a flat space is very likely); this is reflected in Figure 5 (top two). Alternatively, the maps sampled without the likelihood constraint have more obstacles, as seen in Figure 5 (third and fourth). Lastly, notice that the maps sampled by $SGMR_D$ are distanced even from the other *SGMR* maps. In [Summerville *et al.*, 2016], the authors note that the player in Video D attempted to collect all the coins and took long paths through the maps. This resulted in maps with many more structures and platforms than the other sampled maps, and could explain the distance from the other sampled maps.

An additional benefit of separating the player movement model from the map model is the ability to examine the movement model independently. Figure 4 shows a visualization of the *Actions Only* player movement model (showing only the two most probable transitions between each action, for clarity). This allows us to investigate what types of behaviors are common or uncommon given our trained model. For example, from the figure, we can see that many of the actions have a high probability of transitioning to the "None" action

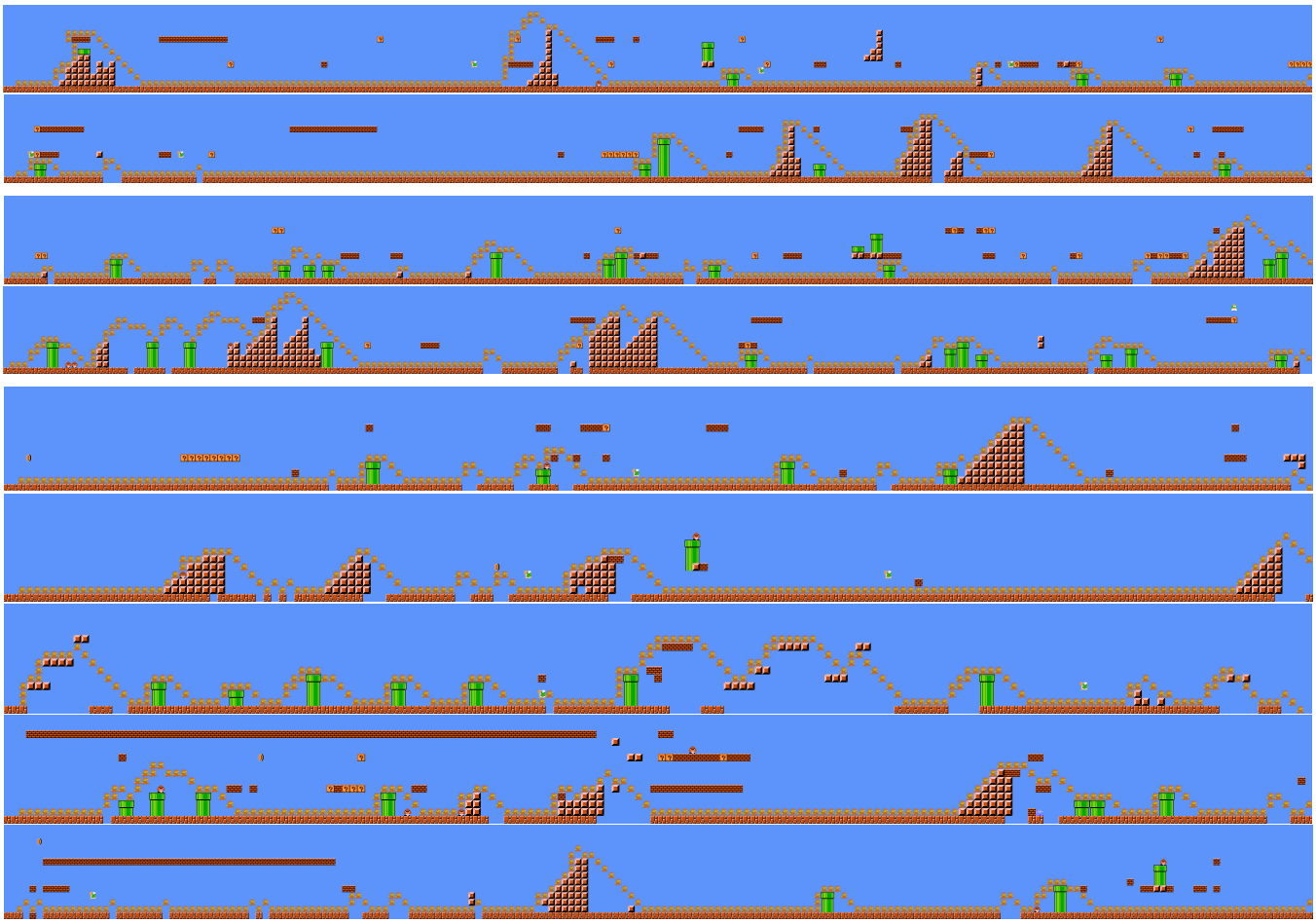

Figure 5: This figure shows example maps sampled using the $VLR_{Likelihood}$ approach (top two), $VLR_{Playability}$ (third and fourth), and an example map for the *SGMR* approach with each of the play traces ($A - D$) and All (bottom five, in order).

(not moving). This indicates that it is likely for the player to change between different actions by first stopping (e.g., if the player is moving left ("L") and wants to move right ("R"), it is more likely for the player to stop moving, and then begin moving right than it is to immediately start moving right).

Figure 5 shows randomly selected sampled maps from each of the approaches (also highlighted in Figure 3). As expected of the $VLR_{Likelihood}$ maps (top 2), there are long stretches of flat terrain, whereas the $VLR_{Playability}$ maps (third and fourth) have many more mountainous structures and pipes.

Our most important result is that we are able to sample maps that force the A* agent to take higher likelihood paths through the maps. This shows that by guiding our sampling algorithm with our player movement model we can sample maps that afford paths that resemble those taken by the human players from whom the player movement model was learned.

6 Conclusions and Future Work

This paper presents an approach for automatically building a player movement model from a gameplay video and using that model to guide a machine learning-based generator to create video game maps that allow for the types of movements observed in the gameplay video. We tested our approach in the domain of *Super Mario Bros.* We found that training the movement model separately from the map generator, we are able to produce maps that are more similar to the training maps than other generators while allowing for high-likelihood paths according to our learned movement model.

In the future, we would like to investigate how training our movement models using different players' gameplay videos may affect the sampled levels. We are also interested in employing more human-like agents in place of the A^* agent, in order to avoid biasing the sampler towards "speed run" style maps. We would like to apply this approach to more complex domains as well, such as *Loderunner* where paths through maps are more complex and require backtracking. Lastly, we noticed that the more likely the path for a generated map, the flatter that map tends to be. This may be because we only constrain our sampling on the conditional probability distribution, and not on other factors, such as the raw distribution of actions in the gameplay trace. We would like to explore this prospect more fully.

References

[Cenkner *et al.*, 2011] Andrew Cenkner, Vadim Bulitko, and Marcia Spetch. A generative computational model for human hide and seek behavior. In *AIIDE*, 2011.

[Ching *et al.*, 2013] Wai-Ki Ching, Ximin Huang, Michael K Ng, and Tak-Kuen Siu. Higher-order markov chains. In *Markov Chains*, pages 141–176. Springer, 2013.

[Dahlskog *et al.*, 2014] Steve Dahlskog, Julian Togelius, and Mark J Nelson. Linear levels through n-grams. *Proceedings of the 18th International Academic MindTrek*, 2014.

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

[Hastings *et al.*, 2009] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. Evolving content in the galactic arms race video game. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 241–248. IEEE, 2009.

[Maaten and Hinton, 2008] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[Markov, 1971] Andrey 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*, pages 552–577. Wiley, 1971.

[Ortega *et al.*, 2013] Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[Park and Jun, 2009] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2):3336–3341, 2009.

[Shaker and Abou-Zleikha, 2014] Noor Shaker and Mohamed Abou-Zleikha. 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*, 2014.

[Shaker *et al.*, 2015] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[Snodgrass and Ontañón, 2014] Sam Snodgrass and Santiago Ontañón. Experiments in map generation using markov chains. In *Proceedings of the 9th International Conference on Foundations of Digital Games*, volume 14, 2014.

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

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

[Summerville and Mateas, 2015] Adam Summerville and Michael Mateas. Sampling hyrule: Sampling probabilistic machine learning for level generation. 2015.

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

[Summerville *et al.*, 2015] Adam James Summerville, Shweta Philip, and Michael Mateas. MCMCTS PCG 4 SMB: Monte Carlo tree search to guide platformer level generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[Summerville *et al.*, 2016] Adam Summerville, Matthew Guzdial, Michael Mateas, and Mark 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.

[Summerville *et al.*, 2017] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *arXiv preprint arXiv:1702.00539*, 2017.

[Togelius *et al.*, 2007] Julian Togelius, Renzo De Nardi, and Simon M Lucas. Towards automatic personalised content creation for racing games. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007.

[Yannakakis and Togelius, 2011] Georgios N Yannakakis and Julian Togelius. Experience-driven procedural content generation. *IEEE Transactions on Affective Computing*, 2(3):147–161, 2011.