Proceedings, The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

An Approach to Domain Transfer in Procedural Content Generation of Two-Dimensional Videogame Levels

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

Abstract

Statistical models, such as Markov Chains, have been recently studied in the context of procedural content generation (PCG). These models can capture statistical regularities of a set of training data and use them to sample new content. However, these techniques assume the existence of sufficient training data with which to train the models. In this paper we study the setting in which we might not have enough training data from the target domain, but we have ample training data from another, similar domain. We propose an algorithm to discover a mapping between domains, so that out-of-domain training data can be used to train the statistical model. Specifically, we apply this to two-dimensional level generation, and experiment with three classic video games: *Super Mario Bros.*, *Kid Icarus* and *Kid Kool*.

Introduction

Procedural content generation (PCG) studies the generation of content (e.g., textures, levels, stories, etc.) algorithmically, allowing players to experience new and unique content. Recent work on machine learning approaches to PCG offers the promise of general PCG algorithms applicable to a wide range of domains (Dahlskog, Togelius, and Nelson 2014; Guzdial and Riedl 2015; Summerville and Mateas 2015). However, these approaches require training data, and a sufficient amount of training data may not be available.

In this paper we present an approach to adapt training data from a source domain to train a procedural content generator for a target domain for which we do not have enough training data. We study this in the context of a PCG approach based on Markov chains. We study how levels from one domain can be translated to levels of another domain via a low-level tile-by-tile mapping, and used as training data in the target domain. We evaluate our approach using three classic videogames: *Super Mario Bros.*, *Kid Icarus*, and *Kid Kool* (two of them being very similar, and one being less similar, in order to study near and far transfer).

The remainder of the paper is organized as follows. First we give an introduction to statistical techniques for level generation, and to domain transfer. Next, we discuss our Markov chain-based level generation approach, and a constrained sampling modification. We then describe our approach for transforming out-of-domain levels into in-domain levels. After that, we describe our experimental set-up. Finally, we discuss our results and draw conclusions.

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Background

Procedural Content Generation. Procedural content generation (PCG) studies the algorithmic creation of content (Shaker, Togelius, and Nelson 2015), typically for video games. Of particular interest to our work are statistical and machine learning approaches to level generation.

For example, Dahlskog et al. proposed sampling new levels using n-grams trained on input levels (Dahlskog, Togelius, and Nelson 2014). Related approaches include generating levels using a statistical model trained on gameplay footage (Guzdial and Riedl 2015), and treating levels as strings and training a recurrent neural network in order to sample new levels (Summerville, Philip, and Mateas 2015). There has also been work on combining multiple machine learning techniques (including Bayes nets and principal component analysis) to sample action RPG levels at multiple levels of detail (Summerville and Mateas 2015).

In our previous work, we proposed techniques that combine statistical sampling with constraint satisfaction via a constrained multi-dimensional Markov chain (MdMC) level generator (Snodgrass and Ontañón 2016).

Domain Transfer. Domain transfer studies adapting knowledge, data, or models from a one domain to be of use in or supplement a related target domain. Domain transfer is a powerful idea that allows an approach to leverage data from outside of its target domain, avoiding the drawbacks of a low amount or low quality of training data. Domain transfer techniques (and domain adaptation, computational analogy, and related techniques) have been explored in the context of cognitive simulation (Falkenhainer, Forbus, and Gentner 1986), where concepts are modeled with predicates and objects and analogies between concepts are found via a search over the mapping of predicates and objects. More recently, domain transfer has been used to supplement the training of classifiers (Duan, Tsang, and Xu 2012; Layne, Hospedales, and Gong 2013) and for transferring textures and styles between images via convolutional neural nets (Yoo et al. 2016; Hertzmann et al. 2001) .

We are interested in developing a domain transfer technique that will allow a statistical level generator to supplement its training data using out-of-domain training data.

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 levels.

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 2014).

Map Representation. A level is represented by an $h \times w$ two-dimensional array, *M*, where *h* is the height of the level, and *w* is the width. Each cell of M is mapped to an element of S, the set of tile types which correspond to the states of the MdMC. We add sentinel tiles to signify the boundaries.

Training. Training an MdMC requires two things: 1) the network structure and 2) training levels. 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 are able to determine the tile configuration (i.e., positions and types of the previous tiles) for a position in the level. 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 as the observed tile frequencies in the training data.

Sampling. Given a desired level size, $h \times w$, a level 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 probability distribution of the trained chain.

While sampling, this method may encounter a tile configuration not seen during training. The probability distribution would thus have not been properly estimated (absolute counts would be 0 for each tile given the previous states). We call this an *unseen state*. Our approach tries to avoid unseen states, since they force the method to generate tiles randomly. We incorporate two strategies to avoid unseen states: *look-ahead* and *fallback*. The look-ahead process samples a fixed number of tiles in advance, trying to ensure that no unseen state is reached. If the look-ahead fails and a tile cannot be found that results in no unseen states, then our method

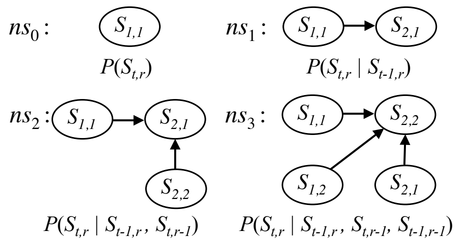

Figure 1: The network structures used in our experiments.

falls back to an MdMC trained with a simpler network structure. In our experiments, we start with network structure ns_3 (shown in Figure 1), which can fall back to ns_2, which itself can fall back to ns_1, and then to ns_0, the raw distribution of tiles in the training levels. More details on this approach can be found in (Snodgrass and Ontañón 2014).

Constraint-based Sampling

Given the approach in this paper will sample levels by transferring data from another domain, it is likely that the resulting levels will not be playable. Thus, the sampling algorithm we propose, in addition to an MdMC, allows the specification of a set of constraints that we want a sampled level to satisfy (such as "making sure the level is playable" or "ensuring a minimum number of enemies"). Specifically, we use a constrained sampling approach called *Violation Location Resampling* (VLR) (Snodgrass and Ontañón 2016). This algorithm (seen in Algorithm 1) takes a width, w, and height, h, of the output level, and a set of constraints, C, and returns a $h \times w$ level satisfying the constraints by using the MdMC approach described above internally. We assume the existence of two functions: $cost(c, Map)$ that checks if a level, Map, satisfies a constraint c or not, and returns the associated cost (0 if the constraint is satisfied, and higher if it is not); and $violations(c, Map)$, which returns a set of sections of the level that violate the constraint. The algorithm works by first sampling a new level (line 1). It then checks if any constraints are violated (line 2). If a constraint is violated, then each section of the level that violates a constraint (line 4) is resampled until its cost is improved (lines 5-18). This is repeated until all constraints are satisfied. For our experiments we use sections of size 12×10.

Domain Transfer

VLR (explained above) is able to sample levels for a target domain when provided with training levels. However, there is no guarantee that the amount of training data available will be enough to train an accurate model. This can result in low quality output levels and a lack of diversity in the levels sampled with the model. To address a lack of training data, we propose an approach that is able to convert levels from other domains into training levels for the target domain.

Tile Mappings. To use levels from one domain to train a model for another domain, the out-of-domain levels need to be translated to the same representation as the in-domain levels. That is, the out-of-domain levels, which use a set of

Algorithm 1 ViolationLocationResampling(w, h, C)

```
1: $Map =$ MdMC([0, 0], [$w, h$])
2: while $\left(\sum_{c \in C} cost(c, Map)\right) > 0$ do
3:   for all $c \in C$ do
4:     for $([x_1, y_1], [x_2, y_2]) \in violations(c, Map)$ do
5:       for all $c_i \in C$ do
6:         $cost_{c_i} = cost(c_i, Map[x_1, y_1][x_2, y_2])$
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 $cost_{c_i} > cost(c_i, m)$ then
12:            GoTo line 9
13:          end if
14:        end for
15:      until $cost(c, m) < cost_c$
16:      $Map[x_1, y_1][x_2, y_2] = m$
17:    end for
18:  end for
19: end while
20: return $Map$
```

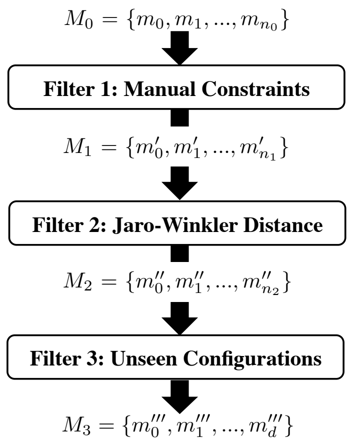
$M_0 = \{m_0, m_1, ..., m_{n_0}\}$
Filter 1: Manual Constraints
$M_1 = \{m'_0, m'_1, ..., m'_{n_1}\}$
Filter 2: Jaro-Winkler Distance
$M_2 = \{m''_0, m''_1, ..., m''_{n_2}\}$
Filter 3: Unseen Configurations
$M_3 = \{m'''_0, m'''_1, ..., m'''_d\}$

Figure 2: The three stage filtering process that we employ to find appropriate tile mappings. We start with the full set of tile mappings, M_0, and reduce this set to the final set of mappings, M_3, by applying three consecutive filters.

tiles, T_{out}, need to use the same set of tile types as the in-domain levels, T_{in}. This is achieved via a *tile mapping*. A tile mapping is function that takes an out-of-domain tile type and returns an in-domain tile type. Formally a tile mapping is defined as a function $m : T_{out}[i] \to T_{in}[j]$.

Moreover, finding an appropriate tile mapping is non-trivial. For example, while empty space tiles are similar in many domains, *Kid Icarus* levels contain moving platform tiles which do not have a direct equivalent in the *Super Mario Bros.* levels we use for training. We propose an approach which automatically defines tile mappings using multi-staged filtering which initially considers the set of all possible mappings that can be defined, and removes undesirable tile mapping with different sieves at each stage.

Searching for Tile Mappings. A sketch of our approach for finding tile mappings from a source to target domain can be seen in Figure 2. Our approach takes the number of desired mappings, d, a threshold, f, for the second filter (Jaro-Winkler), a (possibly empty) set of manual constraints on the tile mappings, a set of in-domain levels, L_{in}, and a set of out-of-domain levels, L_{out}. Our approach returns a set, M_3, of d tile mappings, satisfying the provided constraints.

1. **Filter 1: Manual Constraints**: The first filter in our approach allows the user to define a set of constraints (e.g., “empty tiles should map to empty tiles”, “no more than two out-of-domain tiles should be mapped to the same in-domain-tile”) that she believes should be satisfied in all mappings. Mappings that do not satisfy these constraints are removed from the search space. Note that the user may choose to not define any constraints, in which case the entire search space will be explored. This step allows the user some control over the types of mappings explored.
2. **Filter 2: Jaro-Winkler Distance**: The second filter in our approach automatically reduces the set of tile mappings by comparing the relative frequencies of the tiles in the out-of-domain levels once translated with a mapping, m, to the frequencies of the tiles in the in-domain training levels, and removing those that result in tile frequencies too different from those in the in-domain levels. The idea here is to remove mappings that, intuitively, will lead to poor mappings (e.g., mapping *empty* tiles to *solid* tiles. Specifically, for each remaining mapping, m, in the search space, the levels in L_{out} are converted to in-domain levels using m. Then the tile types are ordered by frequency of appearance in the translated levels, generating a string with each of the tile types in order. We compare this order with the order resulting from sorting the tile types according to their frequency in the in-domain levels using, the Jaro-Winkler distance (Winkler 1999), a distance measure between strings. Each tile mapping whose Jaro-Winkler distance to the in-domain tile ordering is not below a threshold, f, is discarded from the search space.
3. **Filter 3: Unseen Configurations**: The final filter in our approach, filters all but the top d mappings from the current set of mappings by considering the number of *unseen configurations* in the translated out-of-domain levels. An unseen configuration is a combination of tiles that was not observed in the in-domain levels. In order to determine the number of unseen configurations, we first define the window of tiles that makes up a configuration (e.g., 2×2, 3×3). Next, for each configuration appearing in the translated out-of-domain levels, we count the number of times it appears in the in-domain levels. The d mappings with the lowest number of unseen configurations are kept, and returned as the output of this filter. Intuitively, this step trims the search space by removing mappings that have structures that vary greatly from the in-domain levels.

Experimental Evaluation

In order to test our approach, we consider the following scenario: we assume that a single in-domain level is available for training, but a large number of out-of-domain levels are available. We use *Super Mario Bros.* as the target domain, and two games as out-of-domain sources: *Kid Kool*, which is very similar to the target domain, and *Kid Icarus*, which is less similar to the target domain. The technique described in this paper is used to translate the out-of-domain levels to in-domain levels, and use them to train an MdMC model that can then be used to sample new levels using the *Violation Location Resampling* (VLR) algorithm. By choosing one similar and one different domain, we are able to see the varying effects of out-of-domain training data on the model. The remainder of this section describes these domains, experimental set-up and reporting the obtained results.

Super Mario Bros. is a platforming game with linear levels (as defined by (Dahlskog, Togelius, and Nelson 2014)). The player traverses the levels from left to right while avoiding enemies and holes. Our training set contains 16 levels from *Super Mario Bros.* and *Super Mario Bros.: The Lost Levels*. These levels are represented using nine tile types: *S* is a sentinel tile, denoting the borders of a level; *G* represents solid tiles, such as the ground and unbreakable blocks; *B* represents breakable blocks; *?* represents power-up and coin blocks; *p* represents the left section of a pipe; *P* represents the right section of a pipe; *C* represents a bullet bill cannon; *X* represents an enemy; and *E* represents empty space. For our experiments we use the first level in the set that contains all the tile types as our only in-domain training level.

Kid Kool is a platforming game with linear levels, though many levels have multiple sections separated by height (i.e., a sky section that is reachably via platforms, a ground section, and an underground section reachable by some holes in the ground section). Our training set contains 12 levels from *Kid Kool and the Quest for the Seven Wonder Herbs*. These levels are represented using 12 tile types: *S* is a sentinel tile; *G* represents solid tiles; *b* represents collapsing bridges; *B* represents air cannons that blow the player in various directions; *M* represents a spring that allows the player to jump higher when jumped on; *H* represents a tube that can be entered which transports the player to the other end of the same tube; *I* represents a pole that the player can use to launch themselves forward; *W* represents water, which the player can slide across if running, but otherwise is fatal; *T* represents treasure, which can be collected for points; *c* represents cannons which launch bouncing cannonballs; *X* represents enemies; and *E* represents empty space.

Kid Icarus is a platforming game with vertically oriented levels (i.e., the player traverses the levels from bottom to top). Our training set includes six vertically-oriented levels from *Kid Icarus*. These levels are represented using seven tile types: *S* is a sentinel tile; *G* represents solid tiles; *M* represents the entire path of a moving platform; *T* represents a stationary platform (both stationary and moving platforms could be jumped through by a player from the bottom); *H* represents hazards which damage the player; *D* represents a section of a door which leads to various bonus areas; and *E* represents empty space. The levels from this domain are structurally different from the levels in *Super Mario Bros.* and *Kid Kool* because of their vertical orientation.

Table 1: Tile Mappings

Mapping	*Kid Kool*	*Kid Icarus*
Original	**SEGX**`bBMHWTcI`	**SE**`GHMTD`
R^1_1	**SEGX**`?G?PEE?X`	**SE**`GX?B?`
R^1_2	**SEGX**`BXGEP?Pp`	**SE**`BGppp`
R^1_3	**SEGX**`XpXcG?XP`	**SE**`XpBBc`
R^2_1	**SEGX**`?cX?GBEE`	**SE**`E?EBc`
R^2_2	**SEGX**`??XGGBEE`	**SE**`E?PEG`
R^2_3	**SEGX**`XBB?GBEE`	**SE**`cGB?X`
B_1	**SEGX**`XEGEBG?E`	**SE**`pBXcG`
B_2	**SEGX**`XGGEBG?E`	**SE**`PBXcG`
B_3	**SEGX**`XEEEBG?E`	**SE**`?BXcG`
M_1	**SEGX**`?BGpBEEc`	**SE**`GGXGG`
M_2	**SEGX**`GG?pBE?c`	**SE**`BBX?G`
M_3	**SEGX**`GB?PEE?c`	**SE**`BBXGG`

Experimental Setup

In order to evaluate our approach, we compare the tile mappings found using our approach against tile mappings found with other methods. The sets of tile mappings we compare are defined below and can be found in Table 1.

- R^1: chosen at random from the set of tile mappings that satisfy the manual constraints (i.e., from the set of tile mappings after applying the first filter). We chose three such mappings for *Kid Icarus* and *Kid Kool*. They are denoted: R^1_1, R^1_2, and R^1_3.
- R^2: chosen at random from the set of tile mappings remaining after the second filter is applied. We chose three such mappings for *Kid Icarus* and *Kid Kool*. They are denoted: R^2_1, R^2_2, and R^2_3.
- M: defined manually based on our knowledge of the domains. We defined three such mappings for *Kid Icarus* and *Kid Kool*. They are denoted: M_1, M_2, and M_3.
- B: found using our approach with the manual constraints defined below, $d = 3$, and f set to the lowest computed Jaro-Winkler distance among mappings for each domain. Notice, many mappings in our experiments had the same Jaro-Winkler distance. We assumed the existence of a single in-domain training map for this process. These mappings are denoted: B_1, B_2, and B_3.

As mentioned previously, our approach allows the user to define constraints for the first filtering. Below we describe the sets of constraints we used in our experiments:

- *Kid Kool*:
  1. The sentinel tile, *S*, must map to the *Super Mario Bros.* sentinel tile, *S*.
  2. The empty tile, *E*, must map to the empty tile, *E*.
  3. The solid tile, *G*, must map to the solid tile, *G*.
  4. The enemy tile, *X*, must map to the enemy tile, *X*.

Table 2: Comparison of Mappings. *One* corresponds to one map used for training. *All* corresponds to all the original maps in the *Super Mario Bros.* data set, and *BL* is the baseline of training an MdMC with a single training level.

Super Mario Bros. Training Maps			
	Likelihood	Linearity	Leniency
One	-174.00	1.06	0.13
All	-285.69 ± 63.27	2.15 ± 0.85	0.14 ± 0.06
BL	-174.44 ± 15.43	0.93 ± 0.36	0.14 ± 0.03
Kid Kool to *Super Mario Bros.*			
m	Likelihood	Linearity	Leniency
R_1^1	-363.75 ± 40.44	1.44 ± 0.29	$\mathbf{0.14 \pm 0.06}$
R_2^1	-424.50 ± 64.11	1.52 ± 0.30	0.06 ± 0.04
R_3^1	-442.19 ± 47.45	1.50 ± 0.29	0.08 ± 0.04
R_1^2	-412.75 ± 45.83	1.55 ± 0.32	0.07 ± 0.02
R_2^2	-406.38 ± 37.54	1.54 ± 0.25	0.06 ± 0.03
R_3^2	-357.13 ± 52.00	$\mathbf{1.76 \pm 0.34}$	0.06 ± 0.03
B_1	-336.00 ± 27.97	1.49 ± 0.30	0.06 ± 0.02
B_2	-337.50 ± 36.51	1.49 ± 0.27	0.06 ± 0.03
B_3	$\mathbf{-334.63 \pm 45.33}$	1.56 ± 0.29	0.05 ± 0.02
M_1	-337.00 ± 32.69	1.51 ± 0.30	0.15 ± 0.06
M_2	-361.81 ± 63.79	1.53 ± 0.35	$\mathbf{0.14 \pm 0.06}$
M_3	-340.44 ± 46.64	1.63 ± 0.32	0.15 ± 0.06
Kid Icarus to *Super Mario Bros.*			
m	Likelihood	Linearity	Leniency
R_1^1	-906.06 ± 133.75	5.03 ± 0.56	0.11 ± 0.05
R_2^1	-902.63 ± 207.70	$\mathbf{2.59 \pm 1.19}$	0.53 ± 0.17
R_3^1	-1087.62 ± 111.10	4.98 ± 0.65	0.46 ± 0.13
R_1^2	-1042.19 ± 136.87	4.88 ± 0.65	0.42 ± 0.11
R_2^2	-843.69 ± 106.71	5.01 ± 0.57	0.06 ± 0.05
R_3^2	-820.06 ± 135.63	3.02 ± 0.72	0.98 ± 0.22
B_1	-729.31 ± 148.21	5.31 ± 0.72	0.09 ± 0.05
B_2	-681.31 ± 140.16	5.34 ± 0.75	0.09 ± 0.05
B_3	-736.56 ± 125.18	5.28 ± 0.69	0.09 ± 0.06
M_1	$\mathbf{-523.63 \pm 42.68}$	5.14 ± 0.56	$\mathbf{0.13 \pm 0.04}$
M_2	-750.19 ± 81.29	5.39 ± 0.62	0.11 ± 0.05
M_3	-704.13 ± 95.57	5.26 ± 0.65	0.10 ± 0.05

- *Kid Icarus*:

1. The sentinel tile, *S*, must map to the *Super Mario Bros.* sentinel tile, *S*.
2. The empty tile, *E*, must map to the empty tile, *E*.

To evaluate our approach, we configure an MdMC with 12 row splits and a lookahead of 3 and train it using the converted maps and only one training level from *Super Mario Bros.* More details on configuring our MdMC models can be found in (Snodgrass and Ontañón 2014). For our one in-domain level, we chose the first level in the training set that contained all of the tile types. Once trained, we sampled 100 levels of size 12×210 (height $\times$ width) using the VLR algorithm with a single constraint (ensuring playable levels).

We recorded average linearity and leniency of sampled levels (Smith and Whitehead 2010), and average log-likelihood of sampled levels (with a Laplacian smoothed MdMC distribution trained on the single in-domain level).

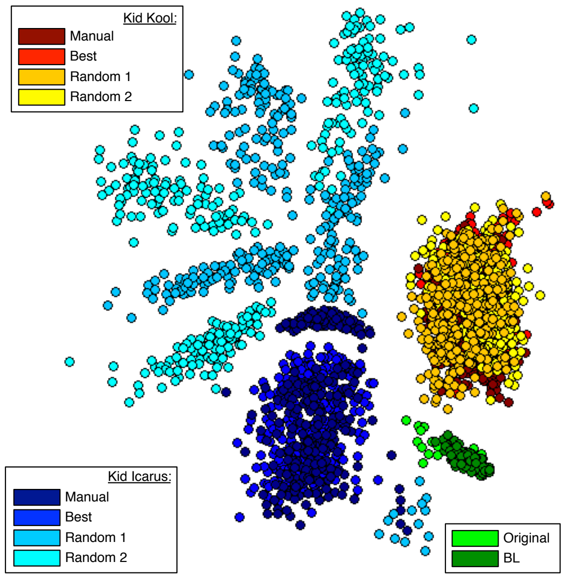

Figure 3: 2-D projection of the original *Super Mario Bros.* levels (green), levels sampled using *Kid Kool* to supplement the training set (yellow and red), and the levels sampled using *Kid Icarus* to supplement the training set (blue).

Results

Table 2 shows the results of our experiments. The top three rows show our metrics applied to the single training level from the *Super Mario Bros.* training set we used, all levels from the *Super Mario Bros.* training set, and levels sampled using an MdMC trained with the single training level. We highlight the evaluation scores most closely matching the set of all training levels for both domains.

The table shows manual mappings and those found by our approach (M_i and B_i) produce levels with higher log-likelihoods than levels sampled using random mappings. This shows that choice of mapping impacts the quality of the levels sampled, and that our method is able to choose quality mappings. Additionally, we see the *Kid Kool* mappings produce levels closer in likelihood to the in-domain training levels than the *Kid Icarus* mappings, which is to be expected since we believe *Kid Kool* is more closely related to *Super Mario Bros.* than *Kid Icarus* is. Further, some random mappings' levels have linearity and leniency similar to the training levels, showing that these mappings capture the general layouts of the levels, but not which tiles should be used to create those layouts (as shown by the lower likelihoods). Furthermore, while the baseline levels closely mirror the single training level, all levels generated by training the MdMC just with one training level are very similar to each other and do not cover the full spectrum of levels. Supplementing the training data with *Kid Kool* mappings found with our method greatly extends the range of levels generated, covering a larger area of the space of levels.

Table 2 also shows much higher linearity values (meaning, less linear levels) for the *Kid Icarus* mappings when compared to the *Kid Kool* mappings and training levels,

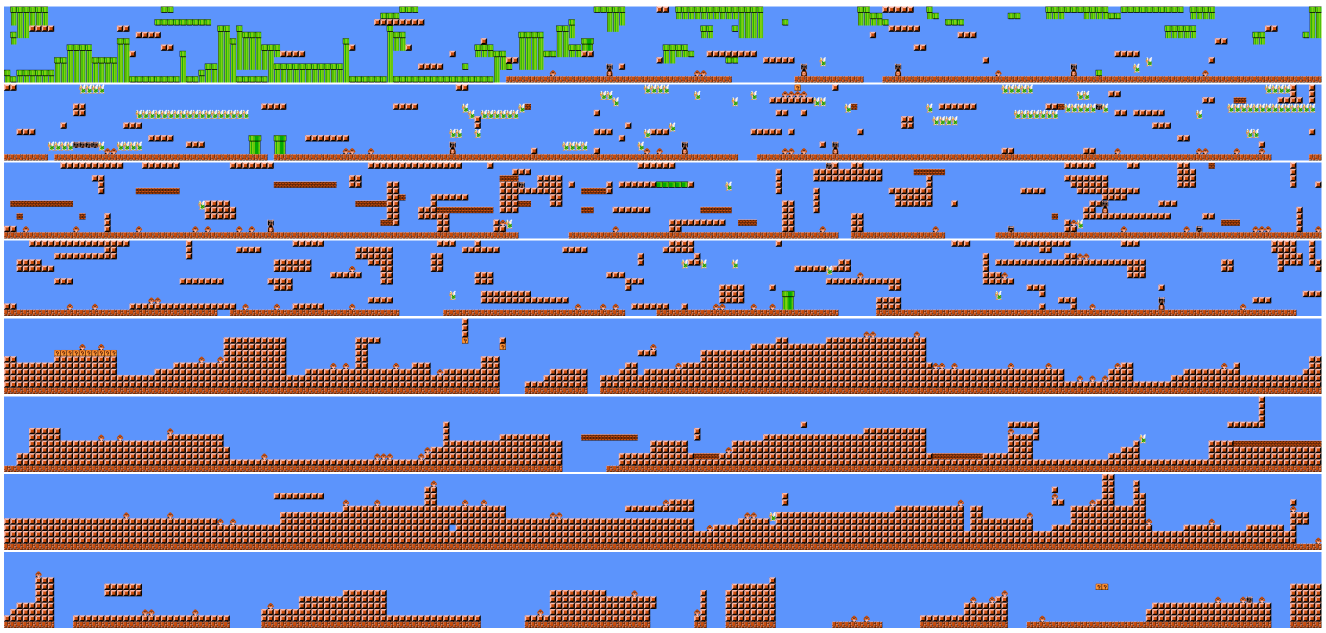

Figure 4: An example level sampled with each mapping method (i.e., manual, random, our approach). From the top, the mappings are *Kid Icarus*: R_2^1, R_3^2, B_2, M_1; *Kid Kool*: R_1^1, R_3^2, B_3, M_3.

showing the effects of the vertical orientation of the *Kid Icarus*. Conversely, the *Kid Kool* mappings have linearity values falling between the training level and the *Kid Icarus* mappings, likely as a result of the mountainous structures and multiple height sections in the *Kid Kool* levels.

Lastly, we see that the manually defined mappings for *Kid Kool* and *Kid Icarus* and the mappings found with our approach for *Kid Icarus* are able to approximate the leniency value of the original training level, whereas the random mappings for *Kid Icarus* vary wildly (due to mappings that assign common tiles to enemies), and are often too low in the remaining *Kid Kool* mappings (due to the vastness of the *Kid Kool* levels paired with the relative infrequency of enemies).

To visualize the space of different levels generated by each approach, Figure 3 shows a two-dimensional projection of the sampled levels along with the 16 *Super Mario Bros.* levels, where each dot represents one level. Levels were projected based on a measure of *distance* between them. To determine the distance between two levels, 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 ($k = 40$) with all the training levels and one transformed level from each tile mapping. It is interesting to see how closely grouped all the levels sampled using the *Kid Kool* levels are (red, yellow, orange). This may be due to our constraints locking more tiles, or due to how similar *Kid Kool* is to *Super Mario Bros.* Additionally, the original *Super Mario Bros.* levels and the levels sampled with the baseline (green) are closer to the *Kid Kool* level clusters than to the *Kid Icarus* level clusters, which further supports that the *Kid Kool* levels are more similar to the training levels. Notice, levels sampled using the *Kid Icarus* (shades of blue) mappings are mostly separated into different clusters corresponding to the different methods (i.e., M_i, R_i^j, B_i). Furthermore, levels produced with the manual mappings (dark blue), are close to the levels produced using the mappings found with our approach (blue), showing that our approach can find mappings more similar to human devised mappings than to random mappings. The figure also shows the narrow space covered by the original levels in the training set.

Figure 4 shows example levels sampled with each tile mapping. The *Kid Icarus* mapping levels contain a large amount of platforms (made of enemies, pipe pieces, and solid tiles), which mimic the structures in the *Kid Icarus* maps. Also note the mountainous structures in the *Kid Kool* mappings levels, which are present in the *Kid Kool* maps.

Conclusions and Future Work

This paper describes an approach for transforming videogame levels from one game to another to supplement a set of training levels in a target domain for use by a statistical procedural level generator. These transformations are done by finding a mapping between the tile types in one game to the tile types in the target domain. Our approach is able to find tile mappings that provide better output than random tile mappings, and similar output to manually produced mappings. Additionally, we find the choice of out-of-domain levels has a large impact on the output levels (i.e., *Kid Icarus* mappings produced levels very different from *Kid Kool* mappings). However, this approach is limited to transferring only what is represented by the current tile format, which does not include gameplay mechanics. In the future, we will explore more complex representations to allow for more complex domains and cross-genre adaptation.

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*.

Duan, L.; Tsang, I. W.; and Xu, D. 2012. Domain transfer multiple kernel learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34(3):465–479.

Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1986. *The structure-mapping engine*. Department of Computer Science, University of Illinois at Urbana-Champaign.

Guzdial, M., and Riedl, M. 2015. Toward game level generation from gameplay videos. In *FDG 2015*.

Hertzmann, A.; Jacobs, C. E.; Oliver, N.; Curless, B.; and Salesin, D. H. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 327–340. ACM.

Layne, R.; Hospedales, T. M.; and Gong, S. 2013. Domain transfer for person re-identification. In *Proceedings of the 4th ACM/IEEE international workshop on Analysis and retrieval of tracked events and motion in imagery stream*, 25–32. ACM.

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.; Togelius, J.; and Nelson, M. J. 2015. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

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

Snodgrass, S., and Ontañón, S. 2014. Experiments in map generation using markov chains. In *FDG 2014*.

Snodgrass, S., and Ontañón, S. 2016. Controllable procedural content generation via constrained multi-dimensional Markov chain sampling. In *Twenty-Fifth International Joint Conference on Artificial Intelligence*.

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. J.; Philip, S.; and Mateas, M. 2015. Mcmcts pcg 4 smb: Monte carlo tree search to guide platformer level generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Winkler, W. E. 1999. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer.

Yoo, D.; Kim, N.; Park, S.; Paek, A. S.; and Kweon, I. S. 2016. Pixel-level domain transfer. *arXiv preprint arXiv:1603.07442*.