Game Level Generation from Gameplay Videos

Matthew Guzdial, Mark Riedl
Entertainment Intelligence Lab
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA, USA
mguzdial3@gatech.edu, riedl@cc.gatech.edu

Abstract

We present an unsupervised process to generate full video game levels from a model trained on gameplay video. The model represents probabilistic relationships between shapes properties, and relates the relationships to stylistic variance within a domain. We utilize the classic platformer game Super Mario Bros. to evaluate this process due to its highly-regarded level design. We evaluate the output in comparison to other data-driven level generation techniques via a user study and demonstrate its ability to produce novel output more stylistically similar to exemplar input.

Introduction

Procedural level generation is the problem of generating high-quality game levels automatically. Existing level generation systems typically employ rule-based methods or formulate optimization problems, where an expert designer encodes domain-specific knowledge as a set of rules, constraints, and/or objective functions. The process of encoding this knowledge is time-consuming and, by its nature, includes the biases of the encoder. This is desirable when the encoder is the game designer, but this is not always the case. As an alternative, data-driven approaches learn a rule set or optimization function from exemplars. However, when it comes to game levels, static exemplars like level maps are insufficient as they lack a record of *how* the level is played.

We propose an alternative to these approaches by learning a generative model from representations of player *experience*: gameplay videos. Given a representation of player experience, a system can learn a generative model of an interactive scene without any additional authored knowledge, as the representation includes how a player interacts with the scene. Building a level design model from gameplay videos allows a system to generate novel levels more in the style of the original levels as the model does not include encoder biases. The system encodes "bias", to an extent, but it is the original expert designer's biases or design style that is encoded rather than those of the algorithm's authors. We therefore advocate for this approach in empowering novice game designers to make game levels for use in education, training, and entertainment, as expert design knowledge is required to either guide novices or fully automate design.

In this paper we present a system that learns a generative, probabilistic model from gameplay video exemplars in an unsupervised fashion. At a high level the system parses raw gameplay video as input, categorizes level sections based on their contents and the player's behavior, and learns a probabilistic graphical model of spatial relationships of content. This model captures the probabilistic structure of level components, which can then be used in the generation of novel content that stylistically matches the original exemplars.

Our contributions include: (1) a method to automatically categorize level data; (2) and a probabilistic model for full level generation. We evaluate our approach by comparing generated levels to those of two other data-driven Super Mario Bros. level generators.

Related Work

Procedural content generation is the umbrella term for systems that take in some design knowledge and output new assets from this knowledge. Approaches include evolutionary search, rule-based systems and instantiating content from probability tables (Hendrikx et al. 2013; Togelius et al. 2011). The often cited goal of these systems is to reduce the authorial burden on designers. However, these systems tend to only transfer the burden from creating content to supplying the design knowledge needed to create that content.

Automatic design knowledge acquisition is the problem of automatically deriving design knowledge from previously extant, high quality exemplars, rather than requiring a direct human author. Dahlskog and Togelius (2014) extracted vertical slices of levels from Super Mario Bros. levels to inform a heuristic function in an evolutionary process. Snodgrass and Ontañón (2014) trained a hierarchical Markov chain on Super Mario Bros. levels, on both tile-by-tile and abstract tile transitions. Hoover et al. (2015) made use of neuroevolution to generate additions to pre-constructed levels. Summerville and Mateas (2016) generated levels using a recurrent neural net trained on tile-to-tile transitions. These approaches require substantial human authoring, such as authored patterns of level content or categorization of stylistically similar level elements (e.g. many different enemies abstracted as "enemy"). In addition, to read in these levels one must either transcribe each level in the game by hand or write code to "strip" the level information from the game.

The problem of generating game levels from gameplay

video is related to object modeling from exemplars, in which a set of exemplar objects are used to train a generative model (Kalogerakis et al. 2014; Fish et al. 2014; Emilien et al. 2015). Our approach builds off these techniques, which also use probabilistic graphical models. However, a majority of object modeling approaches require human-tagging of individual atoms and how they fit together. Our model does not require any human tagging, instead relying on machine vision and probability to smooth any incorrect tags.

Machine vision is not often applied to computer games. However, approaches exist to train agents to play Atari games from pixel input (Mnih et al. 2015; Bellemare et al. 2012). While these approaches and our own use machine vision to process pixels over time, our system focuses on extracting design principles instead of learning to play games.

System Overview

Our system can be understood as containing three parts, operating sequentially. First our system automatically derives sections of level from video and categorizes these sections. Second, the system derives probabilistic graphical models from each category in a process inspired by Kalogerakis et. al (2014). Finally, the system generates novel level sections and combines them based on learned transitions.

We chose to use the classic platformer Super Mario Bros. because of its highly regarded level design and its popularity among the procedural level design community (Shaker et al. 2011; Smith, Whitehead, and Mateas 2010; Sorenson and Pasquier 2010). We begin by supplying our system with two things: a set of videos and a sprite palette. This input is simple to produce with the advent of "Let's Plays" and "Long Plays". By sprite palette we indicate the set of "sprites" or individual images used to build up the levels of a 2D game. For this paper we utilized nine gameplay videos and a fan-authored spritesheet.

Model Learning

Our system learns a generative, probabilistic model of shape-to-shape relationships from gameplay videos. These kinds of models, common in the object modeling field, require a set of similar objects as input. Given that the input to our system is gameplay video, we must determine (1) what input our probabilistic model should learn from and (2) how to categorize this input in an unsupervised fashion to ensure the required similarity. We chose to make use of *level chunks*, sections of level composed of geometric sprite data and learned tags of player information, as the basis of the model. We categorize these level chunks with K-means clustering, and each learned category is then used as input to learn a generative, probabilistic model.

Defining and Categorizing Level Chunks

OpenCV (Pulli et al. 2012), an open machine vision library, allows us to parse our input set of gameplay videos frame-by-frame with the associated spritesheet. The output of this parse is a list of each individual sprite and their positions per frame. From this list, individual frames join together into level chunks when adjacent and if they share 90% of the same content. If a frame differs entirely from the next, its level chunk is marked as the end of a level. Along with a list of sprites and their positions each level chunk stores an *interaction time* value equivalent to the number of frames that combined to form the level chunk. This allows our system to capture how long a player stayed in each level chunk. Interaction time allows our system to do without manually encoded design knowledge representing difficulties or rewards. Instead the interplay between the sprites and interaction time in a chunk allows our system to automatically separate these experiential elements. For example, a level chunk with a large amount of coins and a high interaction time is likely rewarding. With our input of nine gameplay videos this process found 13,492 level chunks.

Our system utilizes K-means clustering to categorize the learned level chunks, with K estimated via the distortion ratio (Pham, Dimov, and Nguyen 2005). We utilize a two-tier clustering approach. For the first round, each level chunk is represented as an n-dimension vector of counts per sprite type (e.g. ground, block, coin) and normalized interaction time. We normalize interaction times for each gameplay video in order to minimize the impact of player skill differences. For the K-means distance metric we utilized Euclidean distance and found twenty-three initial categories.

The first round of clustering allows our system to derive a measure of sprite relevance in the second round of clustering. This makes up for the fact that we do not have design knowledge to determine noteworthy sprites, such as power-up blocks or unusual enemies. We base this measure on a variation of term frequency-inverse document frequency or TF-IDF. TF-IDF is typically utilized in natural language processing and search engines to to determine the relevance of a term in a document. Formally:

$$relevance(t, d, D) = f_{t,d} * log(N/n_t) \quad (1)$$

Where t represents a particular sprite type, d represents a particular level chunk, D represents an entire category of level chunks, N represents the number of level chunks in the category and n_t represents the number of level chunks where the sprite type t appears. $f_{t,d}$ represents the raw number of times a sprite of type t occurs in level chunk d. In this way a notion of relevance can be determined per sprite type per category, which would not be as useful for our system if we used this metric prior to having initial clusters. Along with these relevance scores each level chunk is represented according to its normalized interaction time and a value representing the normalized position of each level chunk in a level. Using the Euclidean distance metric once more for this n-dimensional vector we reclustered each of the categories, finding sixty-nine final clusters, with each first round cluster splitting into an average of three clusters each.

Probabilistic Model

The system builds a probabilistic graphical model from each of the level chunk categories, that represents styles of relative sprite placements. The intuition for this per-category learning is that different types of level chunks will have different relationships, and that therefore different models must

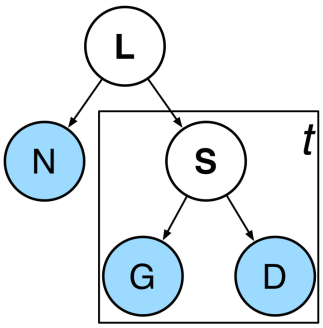

Figure 1: A visualization of the basic probabilistic model.

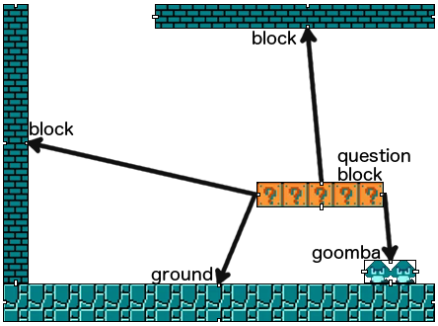

Figure 2: Example of a D Node, the set of relationships.

be learned on an individual category basis. The model extracts values for latent variables to represent probabilistic design rules. Figure 1 shows the probabilistic model using the standard "plate" visualization.. White nodes represent hidden variables, with the blue nodes values learned from the level chunks in a category.

The three observable nodes are the G node, D node, and N node. The G node in the graphical model represents the sprite "geometry", an individual shape of sprite type t. Shapes are built by connecting all adjacent sprites of the same type (e.g. ground, block, coin). Therefore for some types, many of the G nodes are identical, while for others the shapes represent large, irregular patterns of sprites. The D node represents the set of all relative differences between a given G node and all other G nodes in its level chunk. You can see a visual example of this in Figure 2. The D node in this case is the set of vectors capturing relative orientation and direction between the question block shape and all other G nodes in the chunk (two "block" shapes, one "goomba" shape, and one "ground" shape). The N node represents the number of individual atomic sprite values in a particular level chunk. In the case of Figure 2 there are two goombas, seventeen ground sprites, etc.

The first latent variable is the S node, it represents "styles" of sprite shapes. These styles can vary either in geometry or in relative positions. The value and number of S nodes are learned by clustering G and D node pairs with K-means with k estimated according to the distortion ratio. Each pair is a G node, with the corresponding D node representing all connections from that G node to all others in its level chunk. The

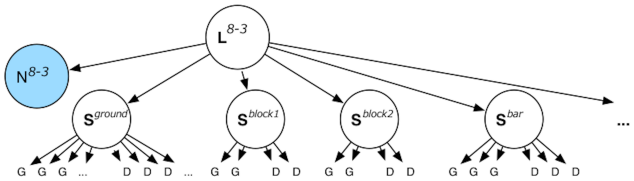

Figure 3: Visualization of a final L Node.

distance function required by K-means clustering weights the G and D parts evenly with G nodes represented as binary matrices undergoing matrix subtraction and D nodes subtracted using Hellman's metric. Clustering with K estimation means the number of S nodes is learned automatically, to best explain the variance in shapes. With a learned S node we can determine the probability of another S node shape at a given relative distance. More formally: $P(g_{s_1}, r_d | g_{s_2})$ or the probability of a G node from *within* a particular S node, given a relative distance to a second G node. For example, goomba shapes have a high probability of co-occurring with ground shapes at the same relative position as in Figure 2.

The L Node represents a specific style of level chunk, the intuition behind it is that it is constituted by the different styles of sprite shapes (S) and the different level chunks that can be built with those shapes (N). The system represents this as a clustering problem, this time of S nodes. Each S node tracks the N node values that arose from the same chunk as its G and D nodes. Essentially, each S node knows the level chunks from the original Mario that represent its "style" of shape. The distance metric between two S nodes is made of two parts. The first part is a normalized value representing the relative size of disjoint set of sprites that co-occur with the S node's style of shape. The second part is a normalized value representing the size of the disjoint set between the N nodes that each S node tracks. This process typically leads to multiple L nodes for a single category of level chunk, in particular it does an excellent job of segregating noisy level chunks that arise from using computer vision techniques.

Figure 3 represents a final learned L Node and all of its children. Notice the multiple S nodes of the "block" type, with one "ground" S node. To create a new level chunk in the "style" of this L node its simply a matter of choosing an N node value and the set of S nodes to constitute it, placing individual shapes in order to maximize their pairwise probabilities. We'll go into this in more detail in the next section.

Generation

Our system generates novel levels in the style of Super Mario Bros. by first instantiating a level "plan" based on learned transitions of level chunk categories, and then by instantiating novel level chunks to fill in that plan. This process is analogous to Joris Dormans' theory of "mission" and "space" generation (Dormans 2010). The level chunk category transitions are derived from the actual gameplay video levels, using the "vocabulary" of learned level chunk categories. These transitions are then used to construct a space of possible level designs where nodes are level chunk cat-

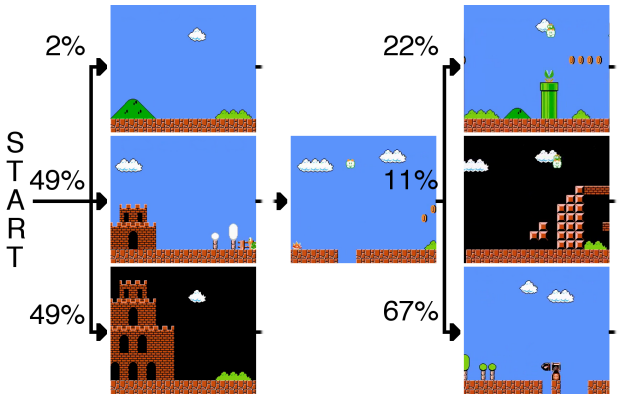

Figure 4: Visualization of a level graph of only overworld levels.

egories and arcs represent the probability of transitioning from one level chunk category to another. A probabilistic walk of the graph can then create a level plan, a sequence of level chunk *categories*. Level chunks can then be generated from the probabilistic model of shape distributions learned for that category. The final level can be imported into a game engine, such as the Infinite Mario engine and played (Togelius, Karakovskiy, and Shaker 2012).

Generating Level Plans

To build up a space of allowed levels our system utilizes the learned level chunk categories to represent the levels from the input videos. During the process of defining level chunks the system automatically discovers the end points of levels, thereby allowing the system to represent levels as sequences of level chunks. From these sequences our system learns a model of transitions between level chunk categories.

Our system uses the fuzzy merge algorithm (Foley 1999) to combine individual sequences of level chunks into a probabilistic, directed graph of level chunk category transitions. This final representation bares a string resemblance to plot graphs (Weyhrauch 1997), which have been shown to be a powerful representation for encoding long sequences of information where absolute ordering is important.

To create the final level graph, each sequence of level chunks is transformed into a linear, directed graph with each level chunk becoming a node. Each node holds its originator's level chunk category, a normalized value representing its position in the level, and its normalized interaction time. Each node is considered in sequence, searching the entirety of the level graph for the best merge point according to a fuzzy definition of equivalence. In this case two nodes are considered equivalent if they have the same category, and their normalized position and interaction time values differ by less than 0.05. When a node is merged, it retains all outward edges, and the weight on each edge is incremented if it matches one of the outward edges in the matched node. If no merge can be found, a node is simply added to the model with the single edge to its prior node in the sequence. This final level graph can be used to generate each of the original levels and novel levels based on taking alternate routes.

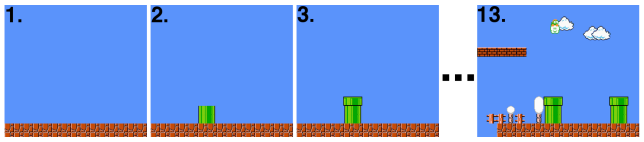

Figure 5: Visualization of a level chunk generation process.

Figure 4 visualizes sections of the level graph learned for "overworld" levels, with each node represented by the median level chunk in that node. Note that the fuzzy merge algorithm correctly learns that these levels almost always start with a big or small castle.

With the level graph, our system is able to construct a "level plan" via a probabilistic walk until it hits a node with no further edges. The chunks of the level plan are then instantiated by generating new level chunks of the types specified by the plan.

Generating Level Chunks

The generation of level chunks takes place for each L node individually. The process is a simple greedy search algorithm, maximizing the following scoring function:

$$1/N * \sum_{i=1}^{N} \sum_{j=1}^{N} p(g_i|g_j, g_i - g_j) \quad (2)$$

Where N is equal to the current number of shapes in a level chunk, g_i is the shape at the ith index, g_j is the shape at the jth index, and $g_i - g_j$ is the relative position of g_i from g_j. This is equivalent to the average of the probabilities of each shape in terms of its relative position to every other shape.

The generation process begins with two things: a single shape chosen randomly from the space of possible shapes in an L node, and a random N node value to serve as an end condition. The N nodes hold count data of sprites from the original level chunks in a category. The N node value serves as an end condition by specifying how many of each sprite type a generated chunk needs to be complete. In every step of the generation process, the system creates a list of possible next shapes, and tests each, choosing the one that maximizes its scoring function. These possible next shapes are chosen according to two metrics: (1) shapes that are still needed to reach the N node value-defined end state and (2) shapes that are required given a shape already in the level chunk. The system defines a shape to require another shape if $p(s_1|s_2) > 0.95$, in other words if the shape styles co-occur more than 95% of the time. We visualize this process in Figure 5, starting with a G node "ground" shape and an N node value set (ground=15, pipeBody=2, pipeTop=1, cloud=2, fence=3, block=10, smallSnowTree=1, and tallSnowTree=1). The shapes not included in the starting N-node come about due to being "required" for some added shape (such as the "lakitu" floating enemy). For more detail on level chunk generation please see (Guzdial and Riedl 2016).

Evaluation

We evaluated our approach with an online user study. The goal of the study was to determine if the our model represented the level design "style" of Super Mario Bros.. We hypothesized that our system would out-perform data-driven level generators on both quantitative and qualitative metrics of style due to the style-encoding probabilistic model.

Experimental Setup

We chose to compare our generator against two other recent data-driven approaches: the Snodgrass and Ontañón, and the Dahlskog and Togelius generators (Dahlskog and Togelius 2014; Snodgrass and Ontañón 2014). We did not include the Summerville and Mateas generator as it was incomplete prior to the human subjects study (Summerville, Philip, and Mateas 2015; Summerville and Mateas 2016).

For each generator we procured five levels to use in our user study, to ensure each level would experience sufficient coverage. We transcribed five randomly selected levels from the nine levels published in (Dahlskog and Togelius 2014). We generated one-hundred random levels from the best-performing version of the Snodgrass generator, and took the first five playable ones. The Snodgrass, and Dahlskog generators were both trained on only the "overworld" levels, and so we modified our generator to only produce "overworld" levels via constraining which level chunk categories it could learn from (Note: our generator is capable or producing other level types, such as "Castle" levels). To deal with the fact that each L node can generate a a varied number of novel level chunks (from 900 to over 400,000), we restricted each L node to only generate one-hundred level chunks. Both other generators required a specified level length, while our system creates levels of arbitrary length. We therefore generated one-hundred levels and selected the five playable levels closest to the specified length. Playability was determined by a greedy-path planning agent. For the study we made use of the Infinite Mario engine (Togelius, Karakovskiy, and Shaker 2012) as the Snodgrass levels were made for this engine. We then translated both the Dahslkog levels and our own automatically after hand-authoring the sprite mappings between Super Mario Bros. and the study engine.

Methodology

Participants played through three levels and answered ten questions: six questions involved ranking the levels they played and four collected demographic information. The first level they played was Super Mario Bros. Level 1-1, translated into the study engine, in order to establish a familiarity with typical Mario levels. Participants were informed that this was an original Mario level. After this point each participant played two levels, one level selected randomly from our generated levels and one level selected randomly from one of the two other generators. Our study therefore had four categories to which participants were randomly assigned, based on which generators' levels they played, and in what order.

The six ranking questions started with asking the participant to rank the two AI levels according to which was the "most like a level from a Mario game". The other five questions asked participants to rank all three played levels according to the following: Fun, Frustration, Challenge, Design, and Creativity. These types of question appear commonly in game surveys (Pedersen, Togelius, and Yannakakis 2009; Drachen et al. 2010). Each question was on its own screen of the study, with the top section showing the very beginning and end of each level the participant played. We chose not to include the entirety of the level as we wanted the participant to make their ranking based on their experience, not on an image representation. After the ranking questions, the participant was asked four demographic questions, which were all Likert-scale style questions. The first three questions were to determine how long ago the participant had played three categories of games: Super Mario Bros., any platformer Mario game, and any platformer game. The last asked how frequently the participants played games in general. We included these questions as we hypothesized that experience with related games may impact a subject's expectations with the generated levels.

Table 1: The results of our system compared to Snodgrass.

	Mario	Ours	Snodgrass	p-value
Mario-like	N/A	1	2	**0.0174**
Fun	1	2	3	**3.02e-5**
Frustration	3	2	1.5	**1.06e-11**
Challenge	3	2	2	0.6976
Design	1	2	3	**2.31e-7**
Deaths	0	2	3	**2.25e-9**

Study Results and Discussion

We recruited seventy-three participants over social media to take part in the study. Participants had to download an application to run the study and e-mail back a zip containing their answers and log files of their playthrough of each level. Due to random assignment we had sixteen individuals in one of our four categories, we therefore made use of sixty-four randomly selected participants' results for statistical tests.

For the first question on how "Mario-like" levels were, we ran the paired Mann-Whitney U test between the two artificial levels. For all other questions we ran Friedman's test between all three level ranking distributions as ranking values are by their nature paired and interrelated. When necessary we made use of the paired Mann-Whitney U tests to ensure individual pairs of generator's rank distributions differed significantly. We split the results according to the "Snodgrass" (Table 1) and "Dahlskog" categories (Table 2). Table 1 and 2 summarize the results. We include the median rank for each generator on each question with 1 as the best and 3 as the worst, and the p-value between the generators' rank distributions. We leave out Creativity, as the distributions were uniform. We also include the median number deaths per level for each generator, and the p-value from running the one way repeated measures ANOVA on each generator's death distributions, recall that there was a maximum of three deaths to each level.

We found in all of the questions that the distributions of

Table 2: The results of our system compared to Dahlskog.

	Mario	Ours	Dahlskog	p-value
Mario-like	N/A	1	2	**0.0383**
Fun	1	2	2	0.3147
Frustration	3	2	1	**3.58e-7**
Challenge	3	2	1	**2.31e-4**
Design	1	2	3	**2.21e-5**
Deaths	0	2	3	**2.42e-5**

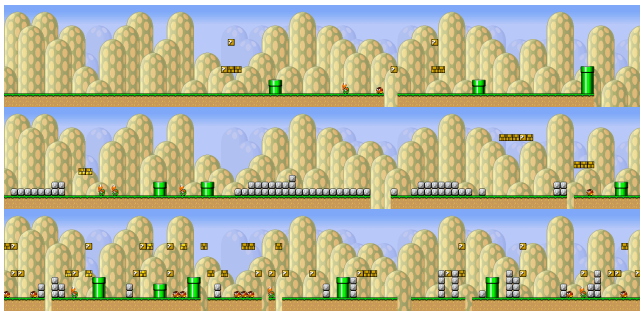

Figure 6: The beginnings of three representative levels from each generator. Top: Our System, Middle: Snodgrass, and Bottom: Dahlskog.

rankings differed significantly, except in the case of Challenge, and Fun, and Creativity. For Creativity all the distributions were uniform, likely due to the lack of a wide-spread definition of level design creativity. These results reflect favorably on our hypothesis that our approach better encapsulated the "style" of Super Mario Bros. levels, in particular that our generator was ranked more highly than both other generators on Mario-likeness and that its Design was ranked second to Level 1-1.

We found no correlation between any of the rankings and the order in which they were presented. We did find a number of correlations between the demographic information and the rankings using Spearman's Rank-Order Correlation. For participants who played Dahlskog levels, we found a strong positive correlation between the time since the participant last played a Mario platformer and how highly they ranked the Dahlskog generator on "Fun" ($r_s = 0.54, p = 0.00147$). This suggests a reason why the Dahlskog generator and our own did not receive significantly different "Fun" rankings, given that individuals who played Mario platformers less frequently ranked the Dahlskog levels more highly. We conclude that Dahlskog levels were "fun" in a different way to typical Mario platformers, due likely to the hand-annotated level design knowledge encoded by the system's authors. This matches our own intuition. As seen in Figure 6, Dahlskog levels contained more enemies and requires more precision jumps than typical Mario levels. In addition we found moderate correlations between how long since the participant had played any platformer and the number of times they died, and the former and the challenge ranking.

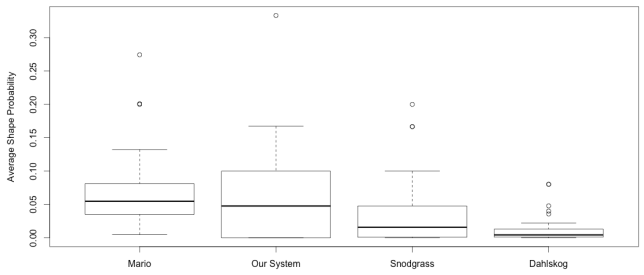

Figure 7: Each generator's boxplot for the model evaluation

Model Evaluation

To confirm that our model actually encodes "style" knowledge correctly we analyzed each set of levels according to the scoring metric presented in Formula 2. This metric captures how likely our model finds a distribution of level elements. Given that the metric is meant for level chunks we "chunked" each level based on screen distance, and matched each chunk to the closest original level chunk to determine what L node to use. We present the results in Figure 7. Our model recognizes the distribution of elements in our generated levels as being more likely than those of the other two generators. We found significant difference between each distribution using the Mann-Whitney U test ($p < 0.05$), and that the median values matched the rankings from the human evaluation. This indicates that our model captures level design style.

Conclusions

We present a probabilistic model of component-based video game level structure that can be use to generate novel video game levels. Our model requires no manual annotation and can extract all knowledge in an unsupervised fashion from gameplay video. Through a user study, we find strong evidence that our model captures style and underlying design of an exemplar set better than the current state of the art.

In the future we hope to utilize this approach on other games and to extract more information from gameplay video. In particular, we believe automatic game mechanic—the rules for how the player and environment can interact—learning from video can provide a better understanding of the semantic meaning of level components. This work comprises a first step toward generalized design knowledge acquisition that can learn from many different games and compose novel game mechanics and level designs.

Acknowledgements

We gratefully thank Sam Snodgrass and Santiago Ontañón for providing source code for their system. This material is based upon work supported by the National Science Foundation under Grant No. 1002748. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. 2012. The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR* abs/1207.4708.

Dahlskog, S., and Togelius, J. 2014. A Multi-level Level Generator. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 1–8. IEEE.

Dormans, J. 2010. Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 1. ACM.

Drachen, A., Nacke, L. E., Yannakakis, G., and Pedersen, A. L. 2010. Correlation Between Heart Rate, Electrodermal Activity and Player Experience in First-person Shooter Games. In *Proceedings of the Fifth ACM SIGGRAPH Symposium on Video Games*, 49–54. ACM.

Emilien, A., Vimont, U., Cani, M.-P., Poulin, P., and Benes, B. 2015. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM Transactions on Graphics* 34(4):106:1–106:11.

Fish, N., Averkiou, M., van Kaick, O., Sorkine-Hornung, O., Cohen-Or, D., and Mitra, N. J. 2014. Meta-representation of Shape Families. *ACM Transactions on Graphics* 33(4):34:1–34:11.

Foley, M. J. 1999. Fuzzy merges: Examples and Techniques. In *Proceedings of the Twenty-Fourth Annual SAS Users Group*.

Guzdial, M., and Riedl, M. 2016. Learning to Blend Computer Game Levels. In *Proceedings of the Seventh International Conference on Computational Creativity (ICCC 2016)*.

Hendrikx, M., Meijer, S., Velden, J. V. D., and Iosup, A. 2013. Procedural Content Generation for Games: A Survey. *ACM Trans. Graph.* 9(1):1:1–1:22.

Hoover, A. K., Togelius, J., and Yannakis, G. N. 2015. Composing Video Game Levels with Music Metaphors through Functional Scaffolding. In *First Computational Creativity and Games Workshop*. ACC.

Kalogerakis, E., Chaudhuri, S., Koller, D., and Koltun, V. 2014. A Probabilistic Model for Component-Based Shape Synthesis. *ACM Transactions on Graphics* 31(4):55:1–55:11.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Pedersen, C., Togelius, J., and Yannakakis, G. N. 2009. Modeling Player Experience in Super Mario Bros. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 132–139. IEEE.

Pham, D. T., Dimov, S. S., and Nguyen, C. D. 2005. Selection of K in K-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 219(1):103–119.

Pulli, K., Baksheev, A., Kornyakov, K., and Eruhimov, V. 2012. Real-Time Computer Vision with OpenCV. *Commun. ACM* 55(6):61–69.

Shaker, N., Togelius, J., Yannakakis, G., Weber, B., Shimizu, T., Hashiyama, T., Sorenson, N., Pasquier, P., Mawhorter, P., Takahashi, G., Smith, G., and Baumgarten, R. 2011. The 2010 Mario AI Championship: Level Generation Track. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(4):332–347.

Smith, G., Whitehead, J., and Mateas, M. 2010. Tanagra: A Mixed-Initiative Level Design Tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, 209–216. New York, NY, USA: ACM.

Snodgrass, S., and Ontañón, S. 2014. Experiments in Map Generation using Markov Chains. *Proceedings of the 9th International Conference on Foundations of Digital Games* 14.

Sorenson, N., and Pasquier, P. 2010. Towards a Generic Framework for Automated Video Game Level Creation. In *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*, EvoApplicatons'10, 131–140. Berlin, Heidelberg: Springer-Verlag.

Summerville, A., and Mateas, M. 2016. Super Mario as a String: Platformer Level Generation via LSTMs. In *Proceedings of the First International Conference of DiGRA and FDG*.

Summerville, A. J., Philip, S., and Mateas, M. 2015. MCMCTS PCG 4 SMB: Monte Carlo Tree Searcg to Guide Platformer Level Generation. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*. AIIDE.

Togelius, J., Yannakakis, G., Stanley, K., and Browne, C. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(3):172–186.

Togelius, J., Karakovskiy, S., and Shaker, N. 2012. 2012 Mario AI Championship. http://www.marioai.org/.

Weyhrauch, P. 1997. *Guiding Interactive Fiction*. Ph.D. Dissertation, Ph. D. Dissertation, Carnegie Mellon University.