Linear levels through n-grams

Steve Dahlskog
Computer Science Department
Malmö University
Malmö, Sweden
steve.dahlskog@mah.se

Julian Togelius, Mark J. Nelson
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
julian@togelius.com, mjas@itu.dk

ABSTRACT

We show that novel, linear game levels can be created using n-grams that have been trained on a corpus of existing levels. The method is fast and simple, and produces levels that are recognisably in the same style as those in the corpus that it has been trained on. We use *Super Mario Bros.* as an example domain, and use a selection of the levels from the original game as a training corpus. We treat Mario levels as a left-to-right sequence of vertical level slices, allowing us to perform level generation in a setting with some formal similarities to n-gram-based text generation and music generation. In empirical results, we investigate the effects of corpus size and n (sequence length). While the applicability of the method might seem limited to the relatively narrow domain of 2D games, we argue that many games in effect have linear levels and n-grams could be used to good effect, given that a suitable alphabet can be found.

Keywords

Procedural content generation, n-grams, videogames

1. INTRODUCTION

Procedural content generation in games (PCG) is the algorithmic creation of game content, either with limited or no human input. Both academia and industry (ranging from AAA-titles to independent productions) have shown interest in PCG in the past few years. PCG has been used to solve numerous content generation problems ranging from runtime level or item generation to design time generation of terrain, game rules or vegetation, using a multitude of different techniques including agents, evolutionary computation, constraint solving, etc. [16].

Recently, participants in a Dagstuhl symposium on artificial and computational intelligence and games proposed a set of long-term goals and research challenges for PCG in a overview paper [15]. These grand goals proposed for PCG are: *Multi-level Multi-content PCG*, *PCG-based Game Design* and *Generating Complete Games*. The paper also proposed nine more concrete research challenges that would support advancement towards reaching the identified grand goals of PCG. Additionally, five concrete actionable steps were proposed, all of which was envisioned to target one or several of the research challenges.

In this paper we look into one of the proposed concrete research challenges, *Representing Style*, and an associated actionable step, *Competent Mario Levels*. Representing style means producing a generative model that follows a particular school of design thinking or a particular designer's recognised style. The Competent Mario Levels actionable step proposes investigating this question by creating level generators for the classic platform game *Super Mario Bros.* (SMB) with the ability to generate varied, interesting, playable, entertaining and good-looking levels.

It is instructive to look at domains other than game content generation to see whether there are methods and ideas that could be brought to bear on a given content generation problem. Are there domains that show similarities to the game domain we are currently investigating, and what techniques do they use? The *n-gram* method has frequently been used to model style and generate novel "randomised" artefacts in two other creative domains, text and music. The n-gram method is very simple – essentially, you build conditional probability tables from strings and sample from these tables when constructing new strings – and also very fast. As simplicity and speed are virtues in PCG, as in so many other domains, it is worth investigating the merits of this method seriously. As far as we know, n-grams have not been used for level generation before (though 2D Markov chains have; see Section 2.3).

We investigate whether we can model the style of the original *SMB* levels by calculating n-gram statistics from those levels, treated as linear sequences of vertical level slices, and then using the resulting Markov level model to produce novel levels that are playable and similar in style to the original levels. As n-grams are used with strings of symbols (such as characters or words, when modelling natural language), we need an "alphabet" for expressing SMB levels as strings. For this purpose, we use a representation of the SMB levels we call "micro-patterns", which are thin vertical slices of a level. In previous work, these slices or micro-patterns were used to create levels where "meso-patterns" and "macro-patterns" decided the order of the slices and therefore gave the levels a structure, style and meaning [3, 4, 5]. Figure 1 shows examples of such slices.

2. CAPTURING PLATFORMER LEVEL STYLE WITH N-GRAMS

Automatically generating levels for *Super-Mario-Bros.*-style platformer games has been studied relatively frequently by procedural content generation researchers [7, 10]. SMB levels have several features that make them tempting to generate automatically. First, they have a quite obvious recurring, pattern-based structure: any player who plays for even a short period of time will start to see similar or even identical patterns of platforms, blocks, and enemy

MindTrek 2014 November 4-7, 2014, Tampere, FINLAND.

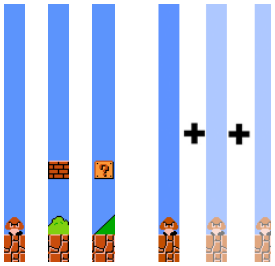


Figure 1: Different slices (micro-patterns) and a Goomba-horde.

placement reappear. Secondly, the levels are typically oriented in a primarily linear, left-to-right direction, so they can be thought of as a sequence of level elements, which is traversed in order.

2.1 N-gram style capture

Several other (otherwise rather different) domains are also characterised by linear sequences that are traversed in order, and exhibit recurring patterns: sequences of notes (music) and sequences of words (language) are two well-known domains in which there has been considerable research into generative methods. An interesting (and early) line of work in such domains has been to model them purely statistically, at a surface level. By surface level what's meant is that models consider only the raw sequences, and don't analyse them in terms of higher-level or semantic structures such as "C major" or "a prepositional phrase". A simple way to do this surface-level statistical modelling is based on counting *n-grams* (n-element subsequences). Given a corpus of writing or a piece of music whose style we want to mimic, we count how often each n-length subsequence appears in the original. We can then produce a new sequence in the same "style" (for a certain definition of style, as we shall discuss) by stringing together n-grams sampled from this bag, weighted according to their original counts[1].

In this paper, we experiment with precisely this kind of n-gram-based generation, but with platformer levels. Our unit is a one-block-wide vertical slice of a level. A complete level is a left-to-right sequence of these vertical slices. This gives us a basic problem formulation very similar to the sequential n-gram-based generation used in music and natural language, allowing borrowing of techniques and cross-domain comparisons.

2.2 Effects in other domains

Since our focus here is *style*, it's worth briefly recounting some typical stylistic effects that n-gram-based generators have in these other domains. In language generation, n-gram generation is most often perceived as a *parody* of a writer's style. Such usage dates back to at least the 1970s [2, item 176], and recurs frequently today, for example in web-based generators that produce n-gram-based mimicry of a Twitter user's updates. Such generators produce a kind of uncanny surface-level reproduction of style: they mimic the sequences of words that a particular writer typically strings together, but when interpreted as sentences or paragraphs, the result is usually nonsense, with little to no interpretable semantic meaning or higher-level structure.

It seems unlikely that level generation will be interpreted in precisely the same way, as a *parody* of a game's levels—though this possibility cannot be ruled out completely. Levels are not typically communicating the same kind of high-level semantic information via their structure as natural language is, and the surface-level style is comparably more important.

A closer comparison may be the case of music. There, the primary complaint has been that n-grams fail to produce interesting high-level structure, at least without being pushed in a way that causes them to lose most of their generativity [9, ch. 3]. With a low n, the music ends up consisting of notes that are locally reasonable, but with an overall piece that wanders in an uninteresting, aimless way, lacking conventional musical features such as movements, recurring themes, loud and quiet periods, perhaps even an interpretable time signature. On the other hand, when n is increased to add larger context to the generator, it soon degenerates into splicing together large preexisting snippets of music—the result of an overfit (insufficiently smoothed) statistical model.

Therefore one of our initial questions to investigate here is whether n-gram generation in *Mario*-style levels results in the same basic problem, of level that are either too wandering, or too cut-and-pasted verbatim from the source material.

2.3 Information content

Viewing platformer levels as sequences of slices leads to interesting analytical and generative possibilities connected to information content and/or entropy of the various sequences. Essentially any sequence of units can be seen as a code transmitting information, and be analysed (with more or less usefulness) using tools from information theory. In fact the first instance of n-gram-based modelling of natural language was performed by Claude Shannon, in the same paper in which he introduced information theory [11].

Later papers used the approach to, for example, statistically characterise the average information carried by each letter of the English alphabet [12]. Use of the information-theory connection to provide control over a generative process also has old roots, dating at least back to 1960s work in computer-music, which used the entropy rate estimated from a Markov model as a tuning knob that the composer could use to vary the entropy of different parts of a piece [6].[2] To our knowledge, similar investigations haven't yet been performed with videogame levels.

This intent to investigate levels as sequences, and thereby gain a close connection to both information theory and previous work in sequence-based Markov modelling in other domains, is also why we don't follow Snodgrass and Ontañón [14] in modelling platformer levels as two-dimensional grids of blocks, and instead use one-dimensional sequences of vertical slices. Two-dimensional extensions of Markov models, such as Markov random fields, have considerably different properties and less of a direct connection to progression over time—though they are indeed interesting to investigate in their own right, and this work is the most closely related to ours, in that it also uses corpus-based statistical modelling to capture level style.

3. METHODS

We represent levels as sequences of vertical level slices (or micropatterns). The full corpus of levels we used for n-gram training is comprised of 15 levels from the original *SMB* game. This includes all levels in the game except for those that have considerably different mechanics from the "normal" ones: water levels, mushroom-platform levels, and boss-fight levels are excluded. In addition, we use the slices within each individual level as "per-

[1]From a more statistical viewpoint, counting n-grams gives us the maximum likelihood estimate of a $(n-1)$th order Markov model presumed to have generated the observed text. Generation then consists of sampling from this Markov model.

[2]For a survey of the uses of Markov modelling for generative music, see [1].

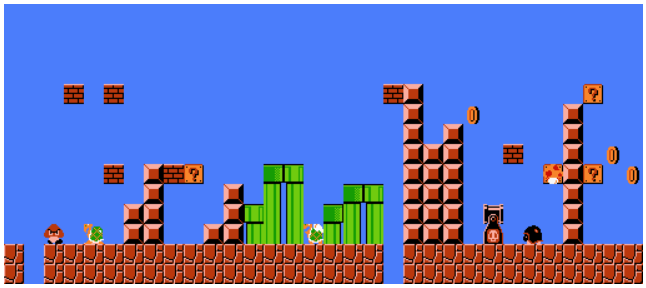


Figure 2: From left to right: the 32 most common slices from the original SMB levels. These slices would therefore be the most frequent unigrams.

level" corpora, in order to investigate whether generators trained on different levels, or combinations of levels, have noticeably different styles.

The original levels vary in length from less than 200 to more than 300 blocks/slices; when generating, we chose a fixed level length of 100. If we incorporate all levels into one single slice-library, several unique slices (seen only once in the game) are found. The largest single addition to the slice library is from level 1–2, due to its "roof" of brick-tiles almost at the top of the screen, which is an unusual arrangement. Combining levels 1–1 and 1–2 more than doubles the slice library from 29 to 73 slices. Figure 2 shows the most common slices that make up the SMB levels.

Our n-gram implementation works by creating separate tables of occurrences for unigrams, bigrams and trigrams. When generating new array (levels), probability tables are calculated based on the occurrence tables. Each new symbol is chosen directly based on its independent probability for unigrams (i.e. the probability for each symbol is exactly its frequency in the original levels). For bigrams, the probability for each symbol is its conditional probability given the preceding character; and for trigrams, its conditional probability given the two previous characters.

There are some special cases. When generating a level using a bigram, the first character will be based on a unigram trained on the same corpus (as there is no preceding character). When using a trigram, the first two characters are based on unigrams. Another special case is for bigrams or (especially) trigrams, when the preceding character or combination of characters has never been followed by anything at all in the corpus. In this case we use a *fallback*: if there is no trigram match for a character combination, we fall back to a bigram, and if there is no bigram, we fall back to a unigram.[3]

4. RESULTS

After initial validation of the functionality of the method, we carried out experiments to investigate the effects of varying n, the effects of varying the training corpus, and to characterise the expressive range of the n-gram generator. We also compared the characteristics of generated levels with those in the initial corpus.

4.1 Effects of varying n

The effects of varying n are rather drastic. Essentially, unigrams produce a haphazard mess, bigrams produce some local structure with much repetition and trigrams produce levels with good local structure that are stylistically similar to the training corpus. In order

[3]This is a fairly simple back-off model, essentially a special case of the widely used Katz back-off model [8], with a back-off threshold of 0, and no discounting.

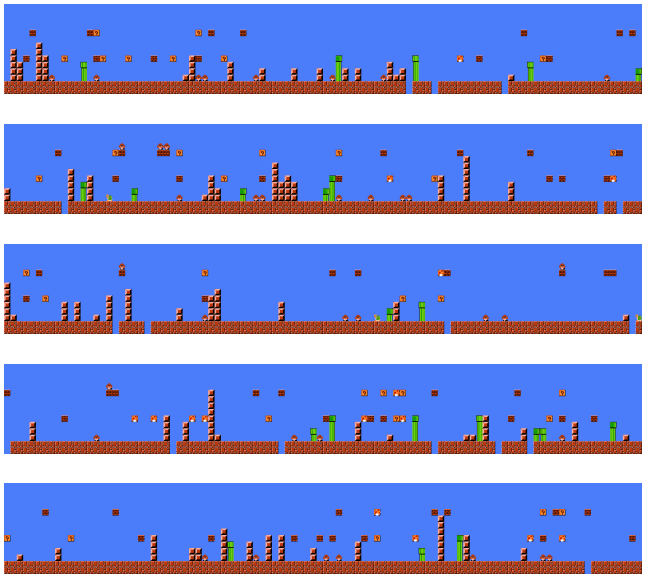


Figure 3: Unigram-based ($n = 1$) levels with SMB World 1–Level 1 as corpus.

to demonstrate these effects we have randomly picked five example levels generated with each configuration.

In figures 3, 4 and 5, we use the same corpus; namely the first level of SMB (199 tiles wide and 14 tiles high) containing 30 different slices. For space saving purposes we have chosen levels of length 100 tiles for all our examples.

In figure 3 we use $n = 1$ for our n-grams, resulting in a rather cluttered level layout with one additional drawback: incorrect pipes. Even though the pipes would be mendable with some rules for the generator, this may need some testing in order to balance the occurrence of pipes. Overall, these levels lack a feel of designed structures, and there is no sense given of imitating any particular style.

After the unigram test we moved on to bigrams ($n = 2$). Figure 4 shows example generated levels, which have working pipes, but also some strange (by SMB standards, that is) "mountain ranges", stairs going both up and down. There are also very few enemies present. Although there seem to be some kind of designed structures, the presence of mountain ranges, however distinct in style, does not convey the style of World 1–Level 1, or of any level in the original SMB. When we increase n to 3, as seen in figure 5, we get correct structures instantiating meso-level design patterns like pipe valleys and stairs (see discussion in [3]), and enemies are also present. Overall, these levels bear a strong stylistic similarity to SMB World 1–Level 1 .

4.2 Effects of varying training data

Of course, SMB contains a lot more content than just World 1–Level 1 (*1–1*). There are 32 levels across 8 worlds, some of which are very different from the others in style, e.g. water levels and boss fight levels. But the appearance of diversity is to some extent deceptive, as several levels are minor but clever variations of others. For instance, levels 1–3 and 5–3 are structurally identical, but with *Bullet Bill*s added to 5–3. Likewise, levels 5–1 and 7–1 are structurally similar, but 7–1 is more *Bullet Bill*-dense. And levels 1–1, 2–1, and 6–2 are all similar, and fairly similar to 1–2 and 4–2.

In order to investigate the results of training n-grams on more

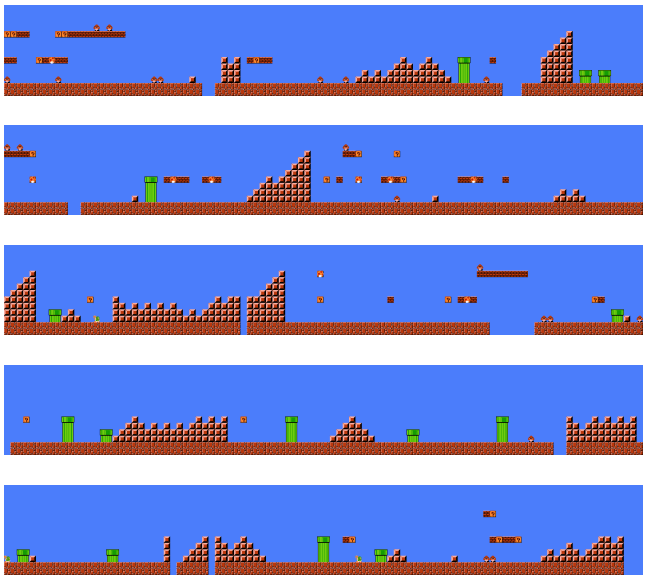


Figure 4: Bigram-based ($n = 2$) levels with SMB world 1–level 1 as corpus.

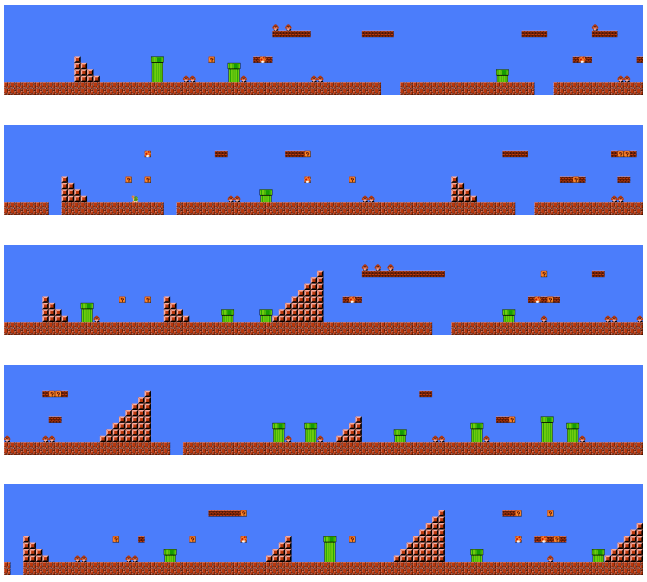


Figure 5: Trigram-based ($n = 3$) levels with SMB 1–1 as corpus.

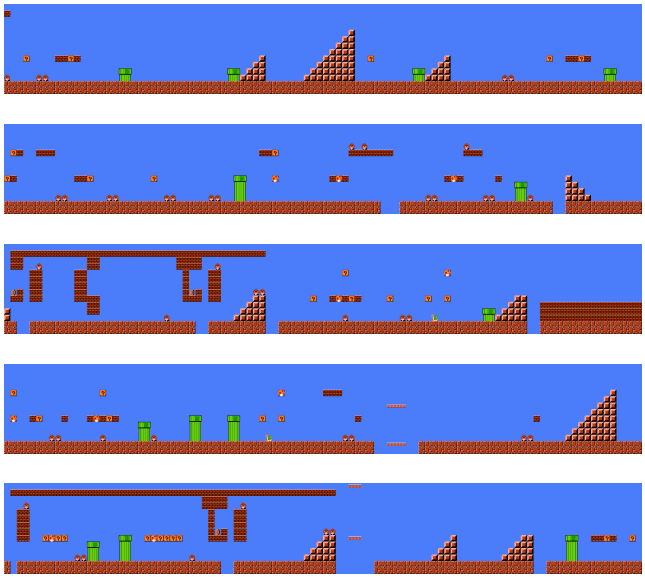


Figure 6: Trigram-based ($n = 3$) levels with SMB 1–1, 1–2 as corpus.

than one level, we created two new corpora: one based on levels 1–1 and 1–2 (see figure 6) and another based on levels 1–1, 1–2 and 2–1 (see figure 7). The combined corpora were created by simply concatenating levels. In figure 6, we can see the effect of training on a corpus consisting of multiple levels in subfigures 3 and 5, where the generated level abruptly changes style. Both begin with the style of level 1–2, and whereas the last one ends in the style of 1–1, the middle one returns to the style of 1–2 after generating a middle section in the style of 1–1.

Continuing with figure 7 we can see that the combination of different levels (e.g. all levels present in the middle and last one) as well as times when the generator sticks to one level style (e.g. the second example). By training on specific levels, the generator follows that particular style for that level, but the larger the corpus, the larger the variation.

As we extended our corpus to include all ground-based levels (no boss, no water and no mushroom/platform levels) each level's traits became part of the corpus and thus influenced the output. Unfortunately the structure of levels becomes clear when incorporating 15 levels (including under-ground levels). Each level starts off with 16 "simple ground" slices with just ground and nothing interesting allowing the player to start in a safe spot. The effect on the generated levels is wide sections of space and nothing interesting from a play perspective. In order to generate interesting levels we shorten these safe-spots so that they do not become to influential in the corpus. We also remove the underground levels for the purpose of preserving the style of surface-levels.

4.3 Expressive range

In order to show the diversity of the method we employ the concept of *expressive range* analysis [7, 13]. Expressive range analysis is a tool for characterising and exploring a PCG method by using a metric; essentially, a number of artifacts (in this case levels) are independently generated, and plotted in the 2-dimensional space of two different metrics. We use the metrics Linearity and Leniency for expressive range comparison (see figure 8 and table 1). A level with a high linearity value forces the player to jump more often

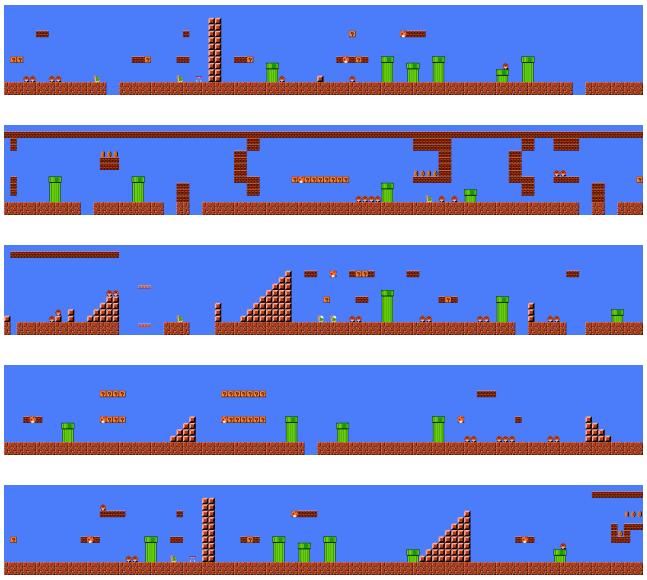
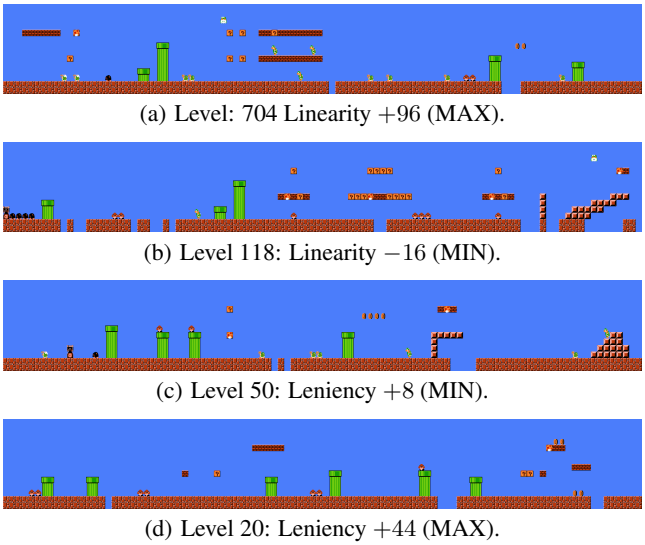


Figure 7: Trigram-based ($n = 3$) levels with SMB 1–1, 1–2 and 2–1 as corpus.



(a) Level: 704 Linearity +96 (MAX).

(b) Level 118: Linearity −16 (MIN).

(c) Level 50: Leniency +8 (MIN).

(d) Level 20: Leniency +44 (MAX).

Figure 8: ($n = 3$) levels with pruned corpus 2600 slices (15 levels from the original SMB with the first screen of each level removed).

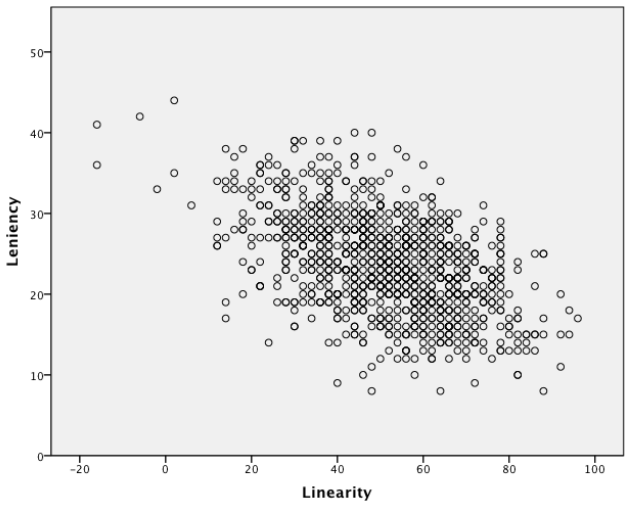


Figure 9: Leniency and Linearity for 1000 above ground pruned levels. Higher Leniency means more difficult. Higher Linearity means flatter levels.

Table 1: Linearity and Leniency.

n-gram callbacks per 1000 levels	252
Linearity Average	51.23
Linearity MIN	-16
Linearity MAX	96
Linearity DEV.	16.79
Leniency Average	23.704
Leniency MIN	8
Leniency MAX	44
Leniency STD.	6.01

than a level with low linearity value. High Leniency means more enemies and gaps where the player may lose a life. An expressive range analysis of 1000 generated levels shows that the output of the n-gram level generator exhibits considerable diversity, at least in these two dimensions.

5. LARGE SCALE COMPARISON

To answer the question of whether this method really allows us to copy style, we did a large scale statistical study of whether generated levels retain the style of those levels that go into their corpus. We chose to use the measures of linearity and leniency, discussed above, as measures of style. If the levels that are generated from a particular corpus have similar measures values for linearity and leniency as those in the corpus, we reason that the levels are similar in style in at least this respect.

We generated 1000 levels based on each original level (only one level in the corpus). We measured the linearity and leniency for each original level and compared that value to the calculated average value for each of the groups of the generated levels. In general, levels generated using n-grams have linearity and leniency values very close to those of the original levels (see Table 2). Exceptions do exist (World 3–Level 1 and 5–1 differ on leniency, and level 1–3 differs on linearity). For level 1–3 the difference may be related to the short original level (length 140), but the other two levels have just below average length. Level 5–1 is on the other hand rather

Table 2: *Lin*earity & *Len*iency comparison between original & average value (1000 generated levels).

Corpus	Lin. (SMB)	Lin. (gen.)	STD (gen.)	Len. (SMB)	Len. (gen.)	STD (gen.)
1-1	26.995	26.486	7.599	-8.295	-8.108	8.481
1-2	19.490	19.277	9.025	10.365	10.241	11.445
1-3	46.030	49.286	16.501	-1.623	-1.429	13.055
2-1	20.152	19.900	5.619	7.082	6.965	7.412
2-3	61.190	60.177	13.822	-35.778	-35.398	18.041
3-1	32.995	33.511	9.012	0.108	-0.532	8.971
3-2	16.385	16.176	7.506	1.455	1.4715	10.538
3-3	45.574	44.286	15.042	-3.983	-3.571	10.548
4-1	15.133	15.640	7.878	16.044	16.588	13.291
4-2	27.411	27.273	11.660	14.318	13.904	14.405
4-3	51.682	51.449	10.336	-22.575	-22.464	9.481
5-1	19.484	19.251	5.345	-1.538	-1.070	8.430
5-2	22.967	22.660	7.258	2.101	1.970	7.547
5-3	43.299	42.857	15.929	-2.731	-2.857	11.812
6-1	31.809	31.609	9.377	-17.194	-17.241	10.516
6-2	34.753	34.597	6.126	20.397	19.905	7.717
6-3	49.930	49.359	8.925	-9.496	-8.974	13.213
7-1	23.768	23.780	7.244	8.372	7.927	9.185
7-3	67.247	66.667	11.575	-40.883	-41.441	19.876
8-1	20.544	20.442	5.922	13.615	13.536	11.656
8-2	16.533	16.019	7.512	17.336	17.476	7.846
8-3	16.273	16.337	8.355	9.288	8.911	6.427

spacious, which may account for the difference in leniency.

The Pearson correlation between linearity of source and generated levels is 0.9985; for leniency, the value is 0.9998. These very strong correlations confirm that the generated levels are indeed very similar to the source levels using this particular measure of style.

6. DISCUSSION

It is clear that several of the configurations of the n-gram level generator, in particular trigrams trained on one level or a few similar levels, generate playable and good-looking levels in the style of those that form the corpus. Thus, the method performs well according to the original criteria (it also takes mere milliseconds to generate a level). It is worth discussing what makes the method work, and how it could be used for other game content generation problems.

6.1 The importance of the representation

The success of the n-gram method in this study is partly because we managed to find a useful set of building blocks for the levels, or in other words a suitable "alphabet". The micro-patterns are few enough to allow meaningful n-gram training even with a small corpus of just a few levels, or in other words strings of just a few hundred characters' length.

A key part of the representation is that the level is seen as a single string, i.e. one-dimensionally. Several other ways of representing an SMB level for n-gram generation would be possible. For example, we could generate each row of blocks separately, with the alphabet consisting of individual blocks and each level consisting of 15 or 20 strings on top of each other. This would have the advantage of a small alphabet, but the considerable disadvantage of the different rows being completely disconnected and the level likely being unplayable. A similar effect would be expected if trying to generate columns rather than rows. While Snodgrass and Ontañón [14] use individual blocks as their alphabet, their use of a 2D Markov chain technique allows this as it takes both horizontal and vertical interactions into account [14].

One could also imagine using longer, larger building blocks, for example sequences that are 5-10 blocks wide, similar to the meso-patterns in our original pattern analysis of SMB levels. However, this would lead to much less perceived variety, as the individual building blocks would be easily identifiable as components of the level.

6.2 Pruning the corpus

In our case the corpus contains several longer sections where the only slice used is the simple ground, since almost all levels in SMB start with 16 slices (a whole screen) of safe area for the player to start in. These longer simple ground sections can skew the n-gram-generation to create uninteresting sections; since n-gram generation has no high-level context, these runs intended to appear at only the beginning of levels can also end up appearing in the middle of them. Similarly, the presence of either extremely common or extremely unique sequences may result in stereotyped structures being simply "copied" from the training corpus recognisably, limiting the variety (in the second example of figure 7, the "c"-like structure appears twice). If the corpus is unbalanced in such manner, we suggest pruning it to reach the desired effect. Alternatively, the generation algorithm could be modified through using a logarithmic transformation on the frequencies, so that less frequent slices are relatively more often chosen (there are many other possible smoothing methods and frequency transformations that can be tried).

6.3 Linearity in game levels

While the method presented here works well for SMB levels, it could be argued that its usefulness is limited to other side-scrolling platformers, or perhaps also similar games such as 2D scrolling shooters (e.g. *R-type*). However, this restriction is not quite as severe as it may appear, if we take into account games whose levels are structurally linear, even if they don't appear as literal left-to-

right side-scrolling sequences. Many games that ostensibly feature 3D worlds with full 3D spatial movement are actually built on linear levels; examples include shooters such as *Halo* and *Call of Duty* (in campaign mode at least), racing games such as *Need for Speed* and *Forza Motorsport* and 3D endless runners such as *Temple Run* and *Canabalt*. Given the identification of a suitable alphabet, the n-gram method could be used as is. Branching paths could be handled by simply generating separate strings for the different paths following a branching point.

7. CONCLUSION

We have shown that n-grams, trained on a corpus consisting of one or several levels from the original *SMB* game, can be used to effectively generate levels that are similar in style to the level(s) used in the corpus. The method is fast and reliable, and gives a reasonable diversity in several dimensions. Using $n = 3$ gives markedly better results than $n = 2$ and particularly compared to $n = 1$ in terms of the visual appeal of the level, and most probably in terms of playability as well. Using a corpus consisting of several levels increases the variety among produced levels, but can lead to surprising shifts in style. It is also found that the generated levels are indeed (on average) very similar to the levels used to learn the n-grams, showed that the method accurately reproduces at least some aspects of style. This simple method has potential to be useful for a large number of games, and should be investigated further in other game domains.

8. REFERENCES

[1] C. Ames. The Markov process as a compositional model: A survey and tutorial. *Leonardo*, 22(2):175–187, 1989.

[2] M. Beeler, R. W. Gosper, and R. Schroeppel. HAKMEM. Technical Report AIM 239, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1972.

[3] S. Dahlskog and J. Togelius. Patterns and procedural content generation: Revisiting Mario in World 1 Level 1. In *Proceedings of the First Workshop on Design Patterns in Games*, 2012.

[4] S. Dahlskog and J. Togelius. Patterns as objectives for level generation. In *Proceedings of the Second Workshop on Design Patterns in Games*, 2013.

[5] S. Dahlskog and J. Togelius. Procedural content generation using patterns as objectives. In *Proceedings of EvoGames, part of EvoStar*, 2014.

[6] L. A. Hiller and R. A. Baker. Computer Cantata: An investigation of compositional procedure. *Perspectives of New Music*, 3:62–90, 1964.

[7] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius. A comparative evaluation of procedural level generators in the Mario AI framework. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*, 2014.

[8] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.

[9] G. Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer, 2009.

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

[11] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[12] C. E. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30(1):50–64, 1951.

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

[14] S. Snodgrass and S. Ontañón. Generating maps using Markov chains. In *Proceedings of the 2013 AIIDE Workshop on Artificial Intelligence and Game Aesthetics*, pages 25–28, 2013.

[15] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley. Procedural content generation: Goals, challenges and actionable steps. In *Dagstuhl Seminar 12191: Artificial and Computational Intelligence in Games*. Dagstuhl, 2013.

[16] J. Togelius, N. Shaker, and M. J. Nelson. Introduction. In N. Shaker, J. Togelius, and M. J. Nelson, editors, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2014.