# Alone We Can Do So Little, Together We Can Do So Much*: A Combinatorial Approach for Generating Game Content

**Noor Shaker**[1] **and Mohamed Abou-Zleikha**[2]
[1]Center for Computer Games and Interaction Design at the IT University of Copenhagen, Copenhagen, Denmark
[2]Department of Electronic Systems, Aalborg University, Aalborg, Denmark
nosh@itu.dk, moa@es.aau.dk

**Abstract**

In this paper we present a procedural content generator using Non-negative Matrix Factorisation (NMF). We use representative levels from five dissimilar content generators to train NMF models that learn patterns about the various components of the game. The constructed models are then used to automatically generate content that resembles the training data as well as to generate novel content through exploring new combinations of patterns. We describe the methodology followed and we show that the generator proposed has a more powerful capability than each of generator taken individually. The generator's output is compared to the other generators using a number of expressivity metrics. The results show that the proposed generator is able to resemble each individual generator as well as demonstrating ability to cover a wider and more novel content space.

## 1 Introduction

Automatic generation of game content is receiving increasing interest among researchers, indie game developers and in the game industry. Several successful applications are presented in the literature motivating further investigations and demonstrating the applicability of this research direction. The first notable implementation of Procedural Content Generation (PCG) is in the game *Rogue* (Toy et al. 1980) where PCG is used to overcome the storage limitation by allowing the creation of complete levels in runtime. Recently, different PCG techniques are employed in several commercial games to provide variations, support human designers, and cut development budget (Shaker, Togelius, and Nelson 2014). Remarkable use-cases include the creature creation interface in *Spore* (Maxis 2008) that allows procedural animations of custom designs, and the random map generation in *Civilization IV* (Firaxis Games 2005). PCG has also witnessed increasing interest among indie game developers mainly for enabling variations (e.g. *Spelunky* (Yu and Hull 2009) and Tiny Wings (Illiger 2011)) boosting creativity (e.g. *Minecraft* (Mojang 2011)) and as assisstive tools used by game designers for automatic play-testing (e.g. *City Conquest* (Intelligence Engine Design Systems LLC 2012)).

A more diverse space of applications is explored by researchers. The focus, however, is mostly given to the automatic generation of individual aspects of game content such as weapons in space shooter games (Hastings, Guha, and Stanley 2009), tracks in car racing games (Cardamone, Loiacono, and Lanzi 2011), opponents in action role-playing games (Blizzard North 1997), level design for physics-based games (Shaker et al. 2013b) and levels in platform games (Shaker et al. 2012; Sorenson and Pasquier 2010). Fewer attempts tackled the problem of evolving complete games, though the successful implementations are still limited to board games (Togelius and Schmidhuber 2008) or simple arcade games (Cook, Colton, and Gow 2012).

The automatic generation of 2D game levels has witnessed notable attention and several studies have been conducted towards realising this goal. Many of the studies examined the popular game *Super Mario Bros* (SMB). The availability of an open source clone of the game (Persson 2009) and the organisation of a competition around this game (Shaker, Togelius, and Yannakakis ) have attracted more attention leading to dozens of papers on this topic ((Sorenson and Pasquier 2010; Shaker et al. 2012; Dahlskog and Togelius 2013) among many others).

Each of the generators proposed for SMB was developed independently and has its distinctive characteristics. While each excels in creating specific types of levels, they are all limited by the design choices imposed by the designers/creators or by the choice of the method. The grammatical evolution generator (Shaker et al. 2012), for instance, use a design grammar and a score to formalise its generative space. Although the use of grammar supports understandability and permits straightforward modifications, once the evolution process starts, the content explored is confined by the final grammar specified and exploring a new space is only possible through a new run after tweaking the grammar. In a recent study (Horn et al. 2014), several dissimilar generators for this game have been analysed and an expressivity analysis experiment is conducted in an attempt to formalise a framework where content generators can be thoroughly investigated and fairly compared. The study showed that although the generators are overall quite different, they all exhibit a limited generative space along the metrics defined. This means that during gameplay, and after playing a number of level, the player becomes aware of the style

*A quote by Helen Keller.

of the generator, can anticipate what will come next and thereafter, the surprise factor will decrease over time resulting in an increase in the state of boredom and ultimately quitting the game. The surprise factor or the presence of a certain degree of unpredictability are important factors for the experience of fun (Davenport et al. 1998; Dormann and Biddle 2006). Variation is a well-known means of exploiting the element of surprise (Skelly 1995) and thereafter in providing an engaging experience (Skelly 1995; Brandtzæg, Følstad, and Heim 2005). Human designers are aware of this issue which is usually avoided through introducing new items, challenges and unexpected events.

When it comes to procedurally generating content, one way to compensate for the surprise factor is to imitate human designers style. Dahlskog et al. (Dahlskog and Togelius 2013) conducted a study in this direction and they proposed an approach that aims specifically at capturing patterns from the original levels of the game designed by human.

In this paper, we aim at boosting variations and therefore supporting unpredictability through combining patterns from different generators. We propose a combinatorial-based approach for content creation using a Non-Negative Matrix Factorization (NMF) method. The use of NMF allows automatic discovery of patterns of arbitrary length, unlike other frequent patterns extraction methods where the length of the patterns to be extracted should be predefined and manually assigned. NMF also permits the possibility of exploring dissimilar variations of a specific pattern through varying its weights.

To apply NMF, a large space of representative content is extracted from five dissimilar generators of SMB. Five independent NMF models capturing different details about content are constructed from all level samples. Each NMF model learns meaningful patterns about a specific game aspect and collectively, the five models can approximate the training data and generate novel content.

## 2 Non-negative Matrix Factorisation

Non-Negative Matrix Factorization (NMF) is a technique for representing non-negative data as a linear parts-based combinations (Lee and Seung 1999). Much of the appeal of NMF comes from its success in learning abstract features from diverse collection of data (Hoyer 2004). The non-negativity property makes NMF results easier to inspect since the representation is purely additive in contrast to many other linear representations such as Principal Component Analysis (PCA) and vector quantisation (VQ) (Lee and Seung 1999; Hoyer 2004). The main difference between these three methods is the qualitative interpretation of the results. While PCA uses eigenvalues, NMF basis are localised features corresponding to parts of the training instances which align better with intuitive notions (Lee and Seung 1999). When the training dataset consisted for instance of a collection of face images, the representation consisted of basis vectors encoding the intuitive face features such as the mouth, nose, eyes, etc (Buchsbaum and Bloch 2002).

Given a non-negative data matrix $\mathbf{V}$, NMF finds an approximate factorization $\mathbf{V} \approx \mathbf{WH}$ where $\mathbf{W}$ and $\mathbf{H}$ are non-negative factors.

Mathematically speaking, NMF is formalised as follows: Given a non-negative data matrix $\mathbf{V} = [v(1), v(2), ..., v(n)] \in \mathbb{R}_{m \times n}$ , NMF decomposes $V$ into the product of two matrices, $\mathbf{W} \in \mathbb{R}_{m \times r}$ and $\mathbf{H} = [h(1), h(2), ..., h(n)] \in \mathbb{R}_{r \times n}$, where all elements in all matrices are exclusively non-negative $w_{ij} \geq 0, v_{ij} \geq, h_{ij} \geq 0, 0 < r < min(n, m)$. The objective function of NMF is to minimize the Euclidean distance between each column of the matrix V and its approximation $\mathbf{V}' = \mathbf{WH}$. This is done as:

$$\mathcal{F}(\mathbf{W}, \mathbf{H}) = \frac{1}{2}||\mathbf{V} - \mathbf{HW}||_F^2 \quad (1)$$

where $||.||_F^2$ denotes the Frobenius norm. Several iterative approaches have been proposed to minimise this criteria such as multiplicative algorithm (Lee and Seung 1999) and an alternating least-squares algorithm (Berry et al. 2007). Each column $v_i$ from the matrix $\mathbf{V}'$ is a linear combination of the columns of the matrix $\mathbf{W}$ which can be seen as parts (i.e. building blocks) of the data. The (r-dimensional) coefficient vector $h_i$ describes how strongly each part is present in the measurement vector $v_i$ such as:

$$v_i = \sum_{l=1}^{r} (w_l * h_i^n(l)) \quad (2)$$

NMF has been successfully applied in many areas to find basis parts (vectors) of faces, clustering of text documents, discovering molecular patterns and speech recognition (Van Hamme 2008; Buchsbaum and Bloch 2002; Brunet et al. 2004; Kim and Tidor 2003). The model has been mostly used for modelling and we are not aware of any previous work that utilised NMF models as generators.

## 3 Experimental Testbed

The testbed game we are using for our experiments is cloned version of *Super Mario Bros*. A number of dissimilar generators have been developed to automatically create content for the game. In this paper, we focus on five of the generators presented in (Horn et al. 2014). These generators were developed independently using different approaches and therefore it is expected that they exhibit different behaviour and that each covers a distinctive expressive space (Horn et al. 2014). In the following section, we give a brief overview of each generator used and the expressivity measures defined to analyse their output spaces.

### Generators

The **Notch** generator is the one originally created for IMB. The generator works by adding components while traversing the level. The component to be added is decided using a number of probability values enabling the creation of endless variation of content with every run.

The **Parametrized** generator takes a number of parameters as input and outputs levels that satisfy these parameters (Shaker, Yannakakis, and Togelius 2013). The version used have the following six input parameters: the number of gaps, width of gaps, number of enemies, enemy placement,

number of power ups and number of boxes. Other aspects of level design are generated at random.

The **Grammatical Evolution (GE)** generator is an evolutionary-based generator that employes a design grammar to specify the structure of levels. Level generation is handled through grammatical evolution and the fitness function optimises for the number of items presented and minimises the conflict in the placement of these items (Shaker et al. 2012).

**Launchpad** is a rhythm-based generator where levels are constructed satisfying rhythmical constraints defined in design grammars. The original generator was modified to generate content compatible with IMB (Smith et al. 2011).

The **Hopper** generator creates content by piecing together small, hand-authored chunks according to predefined probabilities (Shaker et al. 2011).

### Expressivity Metrics

Given a number of dissimilar generators, the questions arise about how the output they create differs? How can we detect and measure these differences? And how can we know which generator is "better"?. Towards finding answers to these questions, attempts have been made to build an expressivity analysis framework in which the output space of several generators can be investigated. Several measures have been proposed (Smith and Whitehead 2010; Shaker et al. 2012; 2013a). In the following, we describe the ones we use to compare the results obtained in our experiments.

The **linearity** of a level is a measure of how flat a level is. A level with a low linearity value exhibits a high number of fluctuations in the height of the platform. To calculate this value, we use the method proposed in (Horn et al. 2014) and we consider the main platform and the hills as causes of deviations of height.

**Leniency** accounts for how easy a level is through counting the number of occurrences of rewarding and harmful items such as coins and enemies. The implementation used in this paper is taken from (Shaker et al. 2012).

A level **Density** is the number of platform stacked on top of each other at each segment. This metric is calculated as described in (Shaker et al. 2012).

## 4 The NMF Generator

Utilising NMF for level generation corresponds to the extraction of the parts (patterns) matrix $W$ from a set of training levels $V$. In our case, these levels are representative outputs from all individual generators combined in one matrix. In order to construct the $V$ matrix, the structure of the levels is converted into sequences of numbers as described in the following section.

### Level Representation

Each level in SMB game is represented as a two dimensional map where each cell contains a game item (a platform, a coin, an enemy, ...). To apply NMF, and to capture as many of the level details, this representation is converted into five basic components as follows:

**Platform, $V_p$:** A vector is generated where each item represents to the height of the basic platform at the corresponding position in the level map.

**Hills, $V_h$:** To accommodate for hills, another vector of the same length is created in which each item stores the number of hills stacked at the corresponding column in the level map. We differentiate between platforms and hills since changes in the platform's height is essential in the design of the game, while the presence of hills, regardless of their height, is what contributes to diversity.

**Gaps, $V_g$:** A vector is used to store whether a column contains a gap. Although $V_g$ can be combined with $V_p$, the analysis showed that better results can be obtained when separated lists are used since this increases the sparsity of the resultant matrix and ultimately improves the overall performance of the NMF model.

**Items, $V_t$:** A vector is created to capture information about the number of rewarding items presented at each column in the level map (coins, blocks and rocks).

**Enemies, $V_e$:** The final vector is binary accounting for the presence of enemies at each column.

As a result, each level is represented by a quintuple $< V_p, V_h, V_g, V_t, V_e >$ capturing various details of the levels.

### Constructing the Model

The final model consists of five independent NMF models constructed for each level component ($NMF_p, NMF_p, NMF_h, NMF_g, NMF_t, NMF_e$).
To estimate these models, each level $i$ in the training set is converted into its corresponding quintuple $< V_{pi}, V_{hi}, V_{go}, V_{ti}, V_{ei} >$ and the results obtained from all levels are arranged in five matrices $V =< \mathbf{V_p}, \mathbf{V_h}, \mathbf{V_g}, \mathbf{V_t}, \mathbf{V_e} >$ that collectively constitute the original levels.

Training the NMF models results in an approximation of the five part matrices $W =< \mathbf{W_p}, \mathbf{W_h}, \mathbf{W_g}, \mathbf{W_t}, \mathbf{W_e} >$ that represent patterns extracted for each component, and five coefficient matrices $H =< \mathbf{H_p}, \mathbf{H_h}, \mathbf{H_g}, \mathbf{H_t}, \mathbf{H_e} >$ corresponding to the weights applied on each pattern. Figure 1 illustrates the overall framework followed.

### Using the Model for Content Generation

Once the model is built and the $Ws$ and $Hs$ matrices are estimated, the model can be used to generate a new level following equation 2. A coefficient vector $h'$ representing the weights applied on the estimated pattern matrices $Ws$ is set as input to the model (see Fig. 1 for illustration). Since the model constitutes five NMFs, reconstructing a single levels corresponds to estimating each of its basis component independently. Consequently, the result is a vector $V' =< \mathbf{V'_p}, \mathbf{V'_h}, \mathbf{V'_g}, \mathbf{V'_t}, \mathbf{V'_e} >$ that represents the reconstructed (or the newly generated, depending on the $h'$ used) level. These vectors are then parsed and composed to create a level in its 2D map format that can be loaded and played.

## 5 Experimental setup

To construct the model, the five generators described previously are used to generate 1000 levels each and the resultant

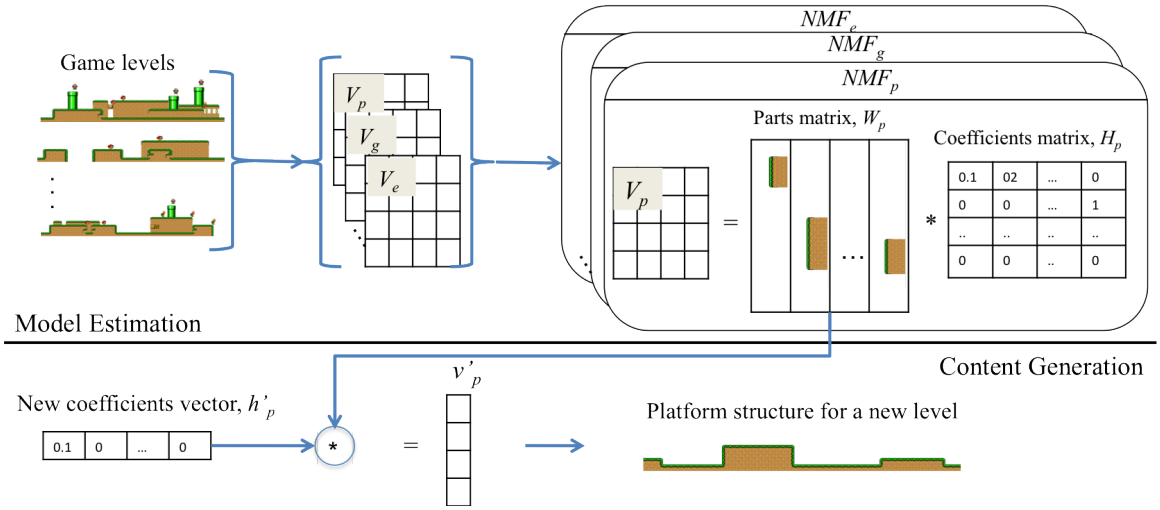


Figure 1: The framework implemented to estimate the model and to use it to generate new content. The upper part shows the steps followed to construct the five NMFs. Levels are first converted into sequences of numbers and arranged in the $V$ matrices that are used as input to estimate $W$ and $H$ for each level component. In the content generation phase (the lower part), and to create a new level, a coefficient vector is used as input that represents the weights applied on the pattern matrix $W$. The result is a vector representing the content of the new level.

set of 5000 levels is used to train the model. Each level map is of size 200 x 15 and converting a level into its basis component quintuple results in five vectors of length 200.

To estimate a single NMF model, a multiplicative update algorithm is used (Lee and Seung 1999). The maximum number of iteration to converge each model is 2000. The number of parts for each component type is experimentally tuned: 40 base vectors were used for platform, 40 for hills, 40 for gaps, 60 for coins and 60 for enemies. The dimensions of the matrices in each model are 200 x 5000, 200 x $B_i$ and $B_i$ x 5000 for $V$, $W$ and $H$, respectively, where $B_i$ is the number of bases vectors for the level component $i$.

## 6 Pattern Analysis

The approach proposed for model construction is used to build a generic content generator. The dataset of 5000 levels is used as input. Five independent NMFs are constructed corresponding to the various level components considered. In the following, we provide a preliminarily analysis on the patterns captured by each NMF model.

Figure 2 presents illustrative patterns captured by the five NMF models constructed for each level component. The patterns extracted by $NMF_p$ represents changes the platform's height and position. $NMF_h$ captures various hill structures such as two or more hills of different width stacked on top of each other. The number of gaps, their width and their spatial placement are captured by $NMF_g$. $NMF_t$ caries information about the different arrangements of coins and blocks. Finally, the ways enemies are scattered around the level are represented in $NMF_e$.

The effect of the coefficient matrix, $H$, on each pattern matrix, $W$, differs according to the component type: $H_p$ has a vertical shifting effect on $W_p$ leading to an increase or decrease in the platform height and the appearance of small platforms on the boundaries (see Fig. 3). $H_h$ and $H_t$

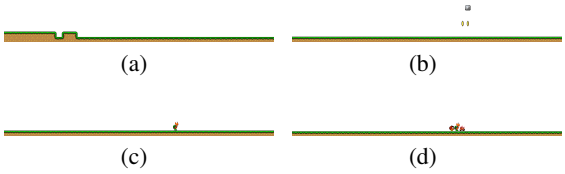


Figure 2: Example patterns extracted by the NMF model from the training data. Subfigures a is extracted by $NMF_p$ and show a platform structure, Subfigure b represents item alignments and are extracted by $NMF_t$ and Subfigures c and d capture various positioning of enemies as captured by $NMF_e$.

shift $W_h$ and $W_t$, respectively along their y-axis resulting in cutting horizontal slices of the patterns; finally, $H_g$ and $H_e$ switch the values of $W_g$ and $W_e$, respectively. Figure 3 presents example patterns from $W_p$ and $W_h$ and the same patterns after applying different coefficient values.

## 7 Experiments and Analysis

We conduct two studies to analyse the efficiency of the generated model and to test its ability to imitate individual generators and to generate novel content. The expressivity measures discussed previously are employed as a framework for comparing the generators' outputs and to provide a quantitative analysis of the results.

### Resembling Individual Generator

In this study, we test the goodness of the generator in mimicking the input generators used for training while maintaining its generalisation capability. To this end, we train the model on the 5000 levels obtained from all generators and we reconstruct these levels from the resultant NMF models.

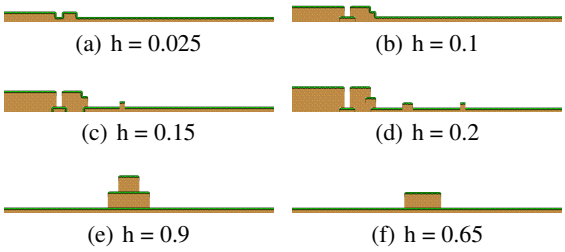


Figure 3: Example patterns of a platform (a) and a hill (f), and the resultant variations after multiplying by different coefficient values.


Figure 4: Example levels generated by Hopper and its corresponding reconstructed level using the NMF model.

This can be done by using the same coefficient matrices estimated by the model during the training process as inputs.

An example level generated by Hopper and its reconstructed level using our model can be seen in Figure 4. To give more thorough insights on the generator's output and check whether the generative space of each generator can be efficiently replicated by our generator, we calculated the values of each of the expressivity metrics for the generated and the original levels. Figure 5 presents the averages and standard deviation values obtained. The results show that our model is capable of effectively resembling the data used for training as indicated by the values obtained from the reconstructed levels which match closely those observed in the training ones for the three expressivity measures.

We also looked at the visualisation of the histogram of the expressive space obtained taking density and leniency as dimensions. Figure 6 presents a pairwise comparison between the ranges obtained by the reconstructed and the original levels along the chosen dimensions. As can be seen, the distributions obtained are very similar. The results, however, show that, in some cases, the original data covers slightly wider expressive ranges compared to the corresponding reconstructed ones. This is anticipated since our model learns the frequently occurring patterns in each generator and not the fine details. This also explains the reason our generator covers mostly the areas of high density. Note however, that it is possible to increase our model's sensitivity and permit grasping more details through the use of higher number of parts (increasing the dimensions of the $W$ and $H$). In the following section, we discuss how our generator is able to cover a wider range when its input is not bounded by the coefficient values of each generator but rather allowed to freely explore the expressive space.

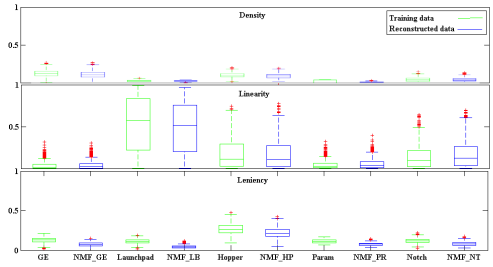


Figure 5: Averages and standard deviation values obtained from levels created by the training generators and the values obtained from the reconstructed levels using our NMF generator for various expressivity metrics.

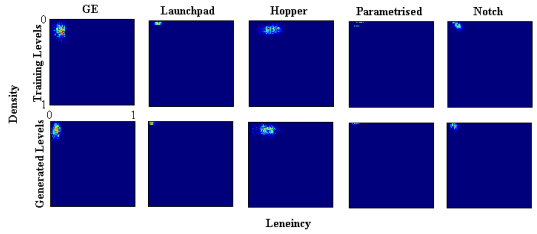


Figure 6: The distribution of 1000 levels obtained from each training generator (upper subfigures) and the corresponding distributions of 1000 levels created by the NMF model when resembling the input levels (lower subfigures). The different colours correspond to the number of levels in each cell; dark blue indicates 0 levels and red corresponds to high level density.

## Exploring Novel Content Space

The novelty of the generator can be tested by analysing its expressive range: the generator is novel if it shows ability to explore areas in the expressive space that were not covered by the generators used for training. This indicates that the generator is able to generalise and improvise new content through combining discovered patterns.

In this study, we investigate whether the proposed generator is able to improvise new levels through combining patterns from different generators. Figure 7 presents the average and standard deviation values obtained for the three expressive measures for the 5000 training levels taken from all generators collectively. The figure also shows the corresponding values for 5000 levels generated by the NMF model when the input coefficient vectors used are the ones estimated by the model and when the model is allowed to freely explore the generative space (using randomly generated coefficient vectors as inputs). The results clearly demonstrate the model's ability to cover wider range than all generators along the density and leniency dimensions. For the linearity metric, and although the average value obtained is very close to the averages from the other settings, the proposed generator shows tendency towards creating highly nonlinear content. This outcome is expected since for an NMF model to create a linear level, a combination of 80 very low coefficient values should be used (40 coefficient for platform $H_p$, and another 40 for hills $H_h$). Although this is possible, it

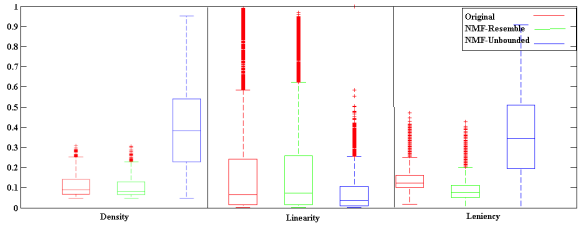


Figure 7: Average and standard deviation values for the three expressive metrics from the 5000 original levels used for training, the levels generated by the NMF model when the coefficient matrix is bounded by the ranges of the original generators and for levels created by the NMF model when the coefficient matrix is unbounded.

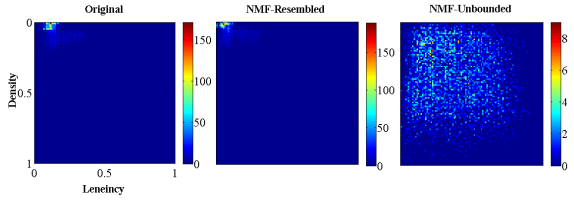


Figure 8: The distribution of 5000 levels created by the generators used for training, by the NMF model when its coefficient matrices are restricted and by the NMF model when it freely explores the expressive space.

is highly unlikely when only 5000 representative levels are generated (which appears to be a slightly small number of levels given the large number of coefficients).

Figure 8 shows the expressive ranges covered along the density and leniency by all generators collectively, and by the levels created by the NMF model when using the estimated $H$ matrices for all generators. The figure also presents the expressive space covered when the model parameters are not constrained. In the former case, similar distributions of levels are obtained while in the latter case, the NMF generator was able to span a significantly wider space on both dimensions with a more even distribution of levels.

Although it is not clear whether covering a wider space is beneficial in terms of player experience or whether the newly explored area is of interest for game designers, the wider space definitely offers more variations and consequently supports creativity by giving designers a greater space of dissimilar patterns to explore. Two representative levels with clear structure variations created by the NMF generator are presented in Figure 9.

## 8 Conclusions

This paper proposes a novel use of the Non-Negative Matrix Factorization method for modelling and generating game content. To train the model, five content generators for the popular game *Super Mario Bros* reported in the literature are used to generate a large and diverse space of content. The levels in the training dataset are converted into sequences from which five NMF models are built where each learns patterns about a specific component of the game. Several experiments are conducted to test the method ability to imitate the style of the input generators and to generate new, and surprising content. This is done through analysing the generative space of each generator along three expressive metrics. The results show that the suggested NMF-based generator is capable of mimicking the input generators used for training as well as demonstrating ability to improvise novel content.

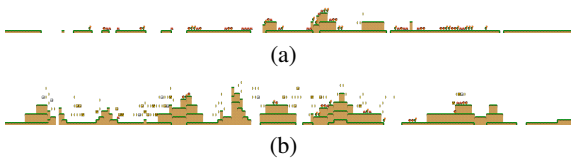


Figure 9: Representative levels generated by the NMF generators when its input space is unbounded.

There are a number of advantages gained by using NMF for content generation: the extraction of patterns of undefined length, the creation of novel content through combining and varying patterns from different generators, the possibility of visualising these patterns and presenting them to game designers who can then select the ones they wish to be present in the final design, and the possibility of training the model on a set of hand-crafted levels to learn the designer style and ultimately use the estimated model for generation. The third point constitutes a future work towards implementing an authoring tool to aid game designers. The latter points motivates further experimentations where the original levels of the game crafted by human designers are used for pattern extraction and style imitation. Future directions will also include investigation of a larger set of expressivity measures along other directions. Example measures include calculating the normalised compression distance between pairs of levels as a direct evaluation of their similarity, another planed experiments is evaluating the novelty of the generated content with human players or designers. Finally, it is worth noting that our model captures patterns across multiple dimensions independently. This can be seen a constraint and strength at the same time since on one hand it restrict the patters extracted to one dimension but on the other hand it permits more variations and freedom in the generation phase when combining patterns opening the possibility for novel content to be explored.

## 9 Acknowledgement

The research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project "PlayGALe" (1337-00172). This work also was supported in part by the Danish Council for Strategic Research of the Danish Agency for Science Technology and Innovation under the CoSound project, case number 11-115328. This publication only reflects the authors views. The authors would like to thank Gillian Smith for granting permission to use the Launchpad generator.

## References

Berry, M. W.; Browne, M.; Langville, A. N.; Pauca, V. P.; and Plemmons, R. J. 2007. Algorithms and applications

for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis* 52(1):155–173.

Blizzard North. 1997. Diablo, Blizzard Entertainment, Ubisoft and Electronic Arts.

Brandtzæg, P. B.; Følstad, A.; and Heim, J. 2005. Enjoyment: lessons from karasek. In *Funology*. Springer. 55–65.

Brunet, J.-P.; Tamayo, P.; Golub, T. R.; and Mesirov, J. P. 2004. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Sciences* 101(12):4164–4169.

Buchsbaum, G., and Bloch, O. 2002. Color categories revealed by non-negative matrix factorization of munsell color spectra. *Vision research* 42(5):559–563.

Cardamone, L.; Loiacono, D.; and Lanzi, P. L. 2011. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 395–402. ACM.

Cook, M.; Colton, S.; and Gow, J. 2012. Initial results from co-operative co-evolution for automated platformer design. *Applications of Evolutionary Computation* 194–203.

Dahlskog, S., and Togelius, J. 2013. Procedural content generation using patterns as objectives.

Davenport, G.; Holmquist, L. E.; Thomas, M.; ; and of Fun Workshop Participants, F. 1998. Fun: A condition of creative research. *IEEE MultiMedia* 5(3):10–15.

Dormann, C., and Biddle, R. 2006. Humour in game-based learning. *Learning, Media and Technology* 31(4):411–424.

Firaxis Games. 2005. Civilization IV, 2K Games & Aspyr.

Hastings, E. J.; Guha, R. K.; and Stanley, K. O. 2009. Evolving content in the galactic arms race video game. In *Proceedings of the 5th international conference on Computational Intelligence and Games*, CIG'09, 241–248. Piscataway, NJ, USA: IEEE Press.

Horn, B.; Dahlskog, S.; Shaker, N.; Smith, G.; and Togelius, J. 2014. A comparative evaluation of procedural level generators in the mario ai framework.

Hoyer, P. O. 2004. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research* 5:1457–1469.

Illiger, A. 2011. Tiny Wings, Andreas Illiger.

Intelligence Engine Design Systems LLC. 2012. City Conquest.

Kim, P. M., and Tidor, B. 2003. Subsystem identification through dimensionality reduction of large-scale gene expression data. *Genome research* 13(7):1706–1718.

Lee, D. D., and Seung, H. S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401(6755):788–791.

Maxis. 2008. Spore, Electronic Arts.

Mojang. 2011. Minecraft, Mojang and Microsoft Studios.

Persson, M. 2009. Infinite mario bros.

Shaker, N.; Togelius, J.; Yannakakis, G. N.; Weber, B.; Shimizu, T.; Hashiyama, T.; Sorenson, N.; Pasquier, P.; Mawhorter, P.; Takahashi, G.; et al. 2011. The 2010 mario ai championship: Level generation track. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(4):332–347.

Shaker, N.; Nicolau, M.; Yannakakis, G.; Togelius, J.; and O?Neill, M. 2012. Evolving levels for super mario bros using grammatical evolution. *IEEE Conference on Computational Intelligence and Games (CIG)* 304–311.

Shaker, M.; Sarhan, M.; Al Naameh, O.; Shaker, N.; and Togelius, J. 2013a. Automatic generation and analysis of physics-based puzzle games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*.

Shaker, M.; Sarhan, M. H.; Naameh, O. A.; Shaker, N.; and Togelius, J. 2013b. Automatic generation and analysis of physics-based puzzle games. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8.

Shaker, N.; Togelius, J.; and Nelson, M. J. 2014. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Shaker, N.; Togelius, J.; and Yannakakis, G. Mario ai championship.

Shaker, N.; Yannakakis, G. N.; and Togelius, J. 2013. Crowdsourcing the aesthetics of platform games. *Computational Intelligence and AI in Games, IEEE Transactions on* 5(3):276–290.

Skelly, T. 1995. Seductive interfaces–engaging, not enraging the user. In *Microsoft Interactive Media Conference*.

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.

Smith, G.; Whitehead, J.; Mateas, M.; Treanor, M.; March, J.; and Cha, M. 2011. Launchpad: A rhythm-based level generator for 2-d platformers. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(1):1–16.

Sorenson, N., and Pasquier, P. 2010. Towards a generic framework for automated video game level creation. In *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, volume 6024, 130–139. Springer LNCS.

Togelius, J., and Schmidhuber, J. 2008. An experiment in automatic game design. In *IEEE Symposium On Computational Intelligence and Games. CIG'08*, 111–118. IEEE.

Toy, M.; Wichman, G.; Arnold, K.; and Lane, J. 1980. Rogue.

Van Hamme, H. 2008. H.: Hac-models: a novel approach to continuous speech recognition. In *Proceedings Interspeech, ISCA*.

Yu, D., and Hull, A. 2009. Spelunky, Independent.