Procedural Content Generation in Games

Ivanov Silviu-Gabriel

Babeș-Bolyai University

May 25, 2018

Abstract

This document explores the field of Machine Learning in the Procedural content generation over the years. This process may be defined as creating machine learning models by using the existing content in games for the purpose of generating new content. During the years the importance of PCG for game development but also for all types of content creators increases, researchers exploring new ways of generating high-quality content, removing the human involvement as much as possible. This paper focuses on using a relatively new paradigm in contrast with constructive , solver-based or search-based methods. In this article I will tackle different methods of procedural content generation via machine learning including: LSTM (long short-term memory networks), deep convolutional networks and autoencoders; clustering; matrix factorization; Markov models: multidimensional Markov chains and n-grams. In the end we will focus on impediments in PCGML like: the lack of data, having to learn from small datasets, style-transfer, multi-layered learning and how PCG functions like a game mechanic.

1.Introduction

Procedural content generation or PCG represents the method of creating game content by using algorithms. During the years it expanded by making use of game development and also technical games research. The value it comes with is represented by reducing the production cost and effort, replay value, saves storage space or creates novel design. An academic PCG researcher takes into consideration this challenge, and also explores how procedural content generation can provides new types of game experience, and also games that can adapt to the player. By building formal models researchers have tackled challenges in computational creativity and also increased our understanding of game design [1]. There are many applications of PCG in the game industry that can be categorized as “constructive” methods, using noise-based algorithms or grammars to create content continuously without evaluation, while others use solver-based methods [2] or search-based methods [3]. All these methods that generate content have in common parameters, constraints, algorithms and objectives created by designers and researchers. Even if we can inspire from the actual game, these AI methods are rarely used to generate new content all by themselves.

Nowadays, a growing interest in the field of machine learning has emerged, and it is represented by the need of creating trained models by using datasets. The most common being the usage of *Deep Neural Network* in *Deep Learning* [4], being used for a large variety of tasks, and also for creating content like: images, videos and audios. The basic idea behind all these algorithms is to create a mode, that is going to be trained on instances of samples coming from a distribution, and after to create new samples using the trained model. This article will focus on the usage of Machine Learning in the field of PCG. The main difference between search-based, solver-based PCG and a procedural content generation using machine learning is the difference in creating the content, while the former algorithms generate new content after a search in the current space, the latter generates directly new content by using the trained model.

The content that is going to be generated can be representing any field from the game, from items, character models, environment to quests, rules and weapons. The models that are going to generate new content can also be of various kinds, from neural networks and decision trees to probabilistic models (Markov models).

Also, the article is going to be focused on the functional part of the game, and not on the design, the artistic part. When we talk about the functional content we refer to the changes that are going to have a great effect on the in-game player experience. One example of content that is going to be excluded is the cosmetics of different objects, because it doesn't directly affect the player actions. We have to mention a key difference between general procedural content generation in all domains and the PCG in games. While in other fields of PCG we are free to create any type of content with little to no constraints, in the field of games we are strictly constrained and limited by the functionalities of the game and his rules – *ergodic media* [5]. A level that have a structure or a number of enemies that lead the player in a dead-end, is not a good one, even if the arrangement of the new content is novel and attractive, it will lead to negative player experience and will be even worse than a level that is finish able but premade and stored from the beginning. Of course, there will be always challenges outside of the games domain that impose all sorts of constraints like creating beautiful lifelike images; however, our focus will be on the constraints and challenges of creating a model content for games.

2. Techniques used in PCGML

We are going to evaluate several Data Representations by using different Training Methods for each one of them.

- **Training Methods**

Here we will utilize several machine learning techniques considered useful for creating and training a model that will be later used in generating in-game content: *Matrix Factorization, Expectation Maximization, Frequency Counting, Back Propagation, Evolution.*

- **Data Representation**

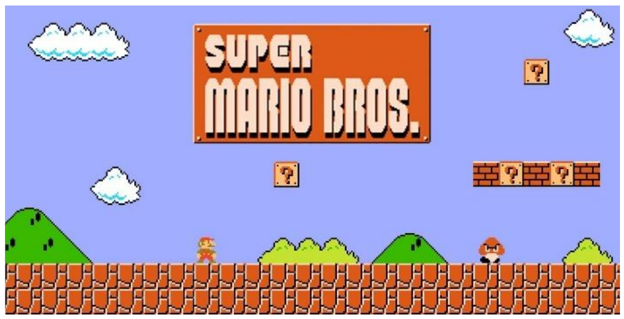
Each level/map/piece of content will be represented in a manner that facilitate the training of the model: *Grids, Sequences* and

Graphs. It is possible for a certain game/type of content to have several representations.

Viewing for each type of representation the techniques that are possibly to be used, will help in comparing them and observe the major difference in terms of improvements and methodology. Also, we will discuss about future work and address gaps in the prior approaches.

a. *Sequences*

Being one of the most popular ways to represent data for content that is going to be experience over time such as textual content or platform game levels. A game that is perfectly fit



for this section is *Super Mario Bros*. Also, other games that feed the content sequentially to the player represent a good choice.

i) Frequency Counting

It refers to methods where the sequence is split into atomic elements calculating the frequency and determining for the next element the generation probability from the current state. The most efficient class of techniques that are able to learn conditional probabilities and are most often used in Machine Learning are the Markov chains. Each state contains multiple previous states by creating the respective *n*-grams of the sequence. One article that focuses

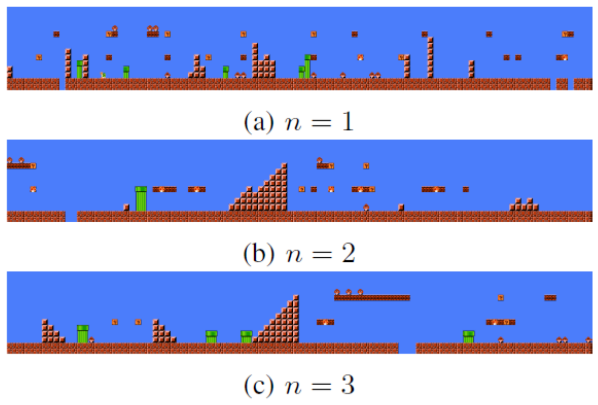


Figure 1: 1,2,3 – grams [6]

on the problem of creating and training a *n*-gram model on the original levels of *Super Mario Bros* is written by Dahlskog et al. and it uses the model to generate new levels [6]. The training process used various levels of *n*, and during the generation we can observe that for 1 – grams we get a random generated level, while 2-grams and 3-grams come close to a real level. See **Figure 1** for examples of this.

ii) Evolution

By using a representation named functional scaffolding for musical composition or FSMC Hoover et al. [7] manages to generate levels for *Super Mario Bros.* To be able to do this, he uses additional voices or types of tiles that are evolved by using ANNs trained. By doing this, he combines the output of the previous iterations with a minimal amount of human-authored content. Evolutionary algorithms that generate content trough machine learning by using fitness functions, thus, the generation is happening trough a provided search space, are not PCGML. One interesting combination of machine learning and search-based algorithms is the DeLeNoX algorithm. It uses unsupervised learning in order to reshape the search space [8].

iii) Back Propagation

For a designer the Artificial Neural Networks represents a universal function approximators being used in the field of PCGML from the beginning. Of course, ANNs can be trained via evolutionary but they are usually being trained by using Back Propagation. Back Propagation referring to the phenomenon of propagating the error during the training phase trough the ANN and changing each of its weights depending on the responsible amount for that error.

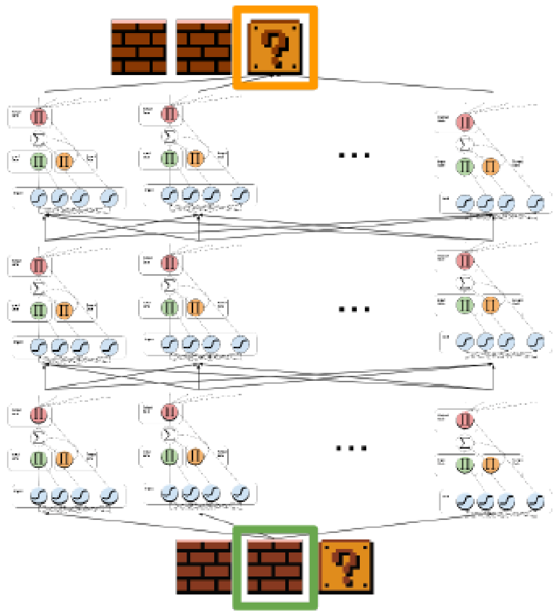
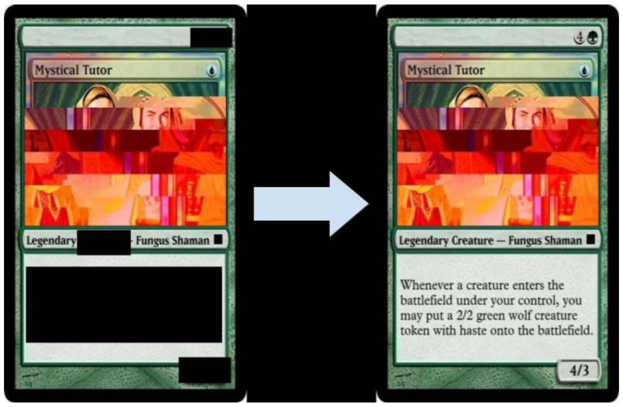


Figure 2: Green represents input tile and orange the output

A Long Short-Term Memory Recurrent Neural Network was implemented for *Super Mario Bros.* by Summerville and Mateas [9]. After a training on a tile representation of each level the model was able to generate new levels. LSTM represents a variety of RNN also representing the current state-of-the-art for parsing sequence-based tasks. The LSTM differ from RNN by the possibility of holding information for up to 100 and 1000-time steps. In their article they processed the sequences representing the map in 3 different ways in order to experiment and to find the best solution. In the end they managed to add information about how deep into the sequence the level geometry was, managing to learn the model not only the basic structure of the level but also level progression and where should be the level ending. See **Figure 2** for an example of LSTM generating a new tile.



Whenever a creature enters the battlefield under your control, you may put a 2/2 green wolf creature token with haste onto the battlefield.

Figure 3: Partial card specification fallowed by the output.

Another distinct example of game content generation using LSTM is represented by *Magic: The Gathering* card generation. The model created by Morgan Milewicz after being trained on a whole set of Magic cards set, it was able to generate new cards [10]. The cards were represented by text fields of Cost, Type, Name, etc. He mentions that a disadvantage of using this representation was the impossibility of conditioning the cards content on the fields that appears latter in the cards sequence, but maybe it would be possible if we start the sequence from the end of the cards and not from the beginning. See **Figure 3** for an example of generation for cards.

b. *Grids*

The majority of the game levels from the 2D era can be represented as two-dimensional grids. Some times the representation comes across some difficulties because of the shape and aligning, but in at the same time this representation is the most natural for many different kinds of levels like dungeons, real time strategy maps and even platformers.

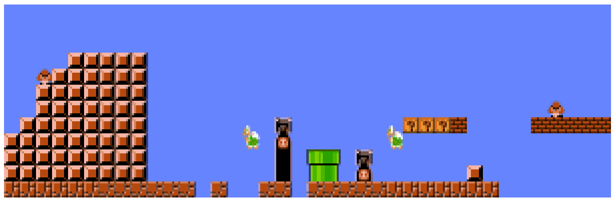


Figure 4: *Super Mario Bros* level generated by MdMCs

i) Frequency Counting

After discussing the one-dimensional Markov chains, the next step is the Multi-dimensional Markov Chains or MdMCs [11]. The difference between the two is that in the one-dimension the state is represented by a single linear dimension while in the Multi-dimensional Markov chains the state represents a surrounding neighborhood., allowing for dependencies from multiple states in multiple directions. Therefore, a new approach to level generation appeared when Snodgrass and Ontañón used in their research a MdMCs. Thus, they started to map again a *Super Mario Bros.* map to a grid and for each tile in the grid the model calculates the probabilities for its type. This time the model is taking into consideration the whole surrounding and not just the previous state. After their new approach, they also devolved a Markov random field or MRF approach which was able to perform better than he standard MdMC model (in a *Kid Icarus* level) [12].

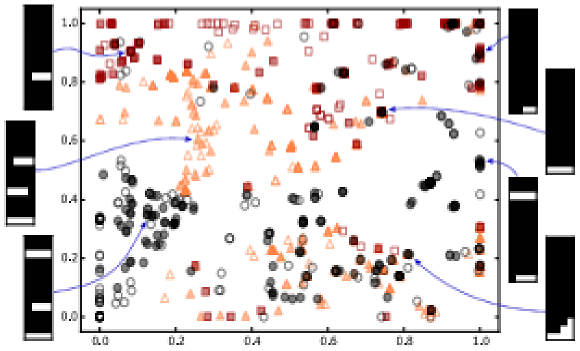
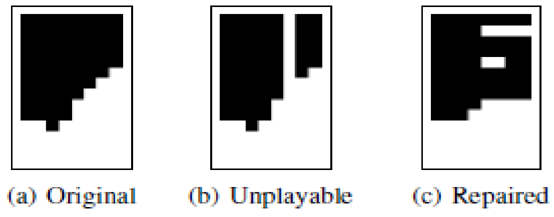


Figure 5: *Super Mario Bros.* Autoencoder detecting the level in the map

ii) Back Propagation

Jan et al. showed how autoencoders are able to be trained to reproduce levels from the original *Super Maria Bros.* game. The autoencoders were trained on vertical level windows from the original levels and transforming them into a more general representation, also they discovered that a 4-tile window is the best dimension for the autoencoder to be trained with [13]. The autoencoder can letter be used to differentiate the generated levels from the original ones, and also to repair the unplayable levels, by changing the tile-level features to make the level playable. This approach is very useful after the process of generating a level using PCG (not only with



Machine Learning) because we are able to repair the levels if they are broken. See **Figure 5** for an example of autolacalization for the current window in the autoencoder without human interaction.

Another usage of Neural Networks comes from Lee et al. [14] who uses convolutional neural networks in order to predict the locations of the resources in *StarCraft II* maps. Usually the resources are placed in order to fallow the topology of the map, it represents a major factor in the gameplay. Using this fact, they transformed the maps in grids and inputs for the training phase of the deep neural networks in order to be able to predict the resource locations. O major challenge for them was represented by the small dataset who often leaded to the overfitting of the neural network. But, by adding some steps at the end of the process and high-quality product emerged allowing the designers to modify and create the resource location without losing the credibility of the game.

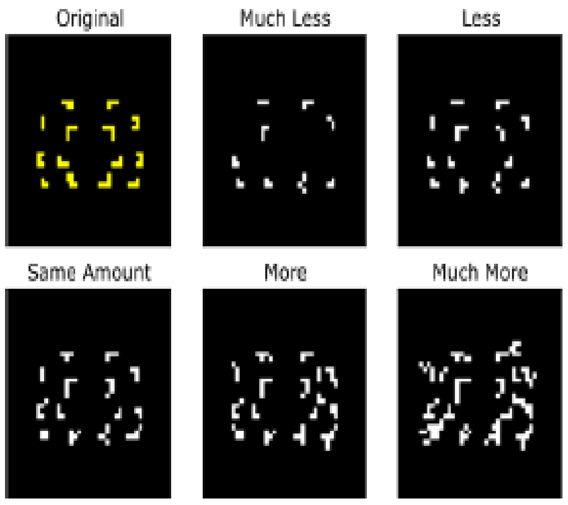


Figure 6: Resources representation on the map, highlighting the frequency

iii) Matrix Factorization

There are some approaches that user matrix factorization to find in high-dimensional data latent level features. In this way features are inferred by compressing the data in smaller matrices. Shaker and Abou-Zleikha managed to create from levels generated with a non-ML-base algorithm, levels that are much more expressive. After the first step, they compress each level into small matrices based on the content type. Each small matrix factorized a transformed into “part matrices” and coefficient matrices, representing the weights for each patter/feature [15].

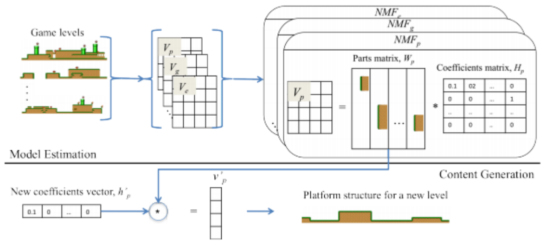


Figure 7: Estimate the model to use is to generate new content

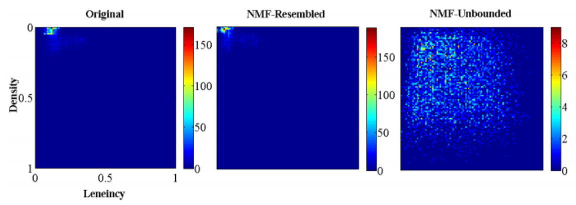


Figure 8: The distribution of 5000 levels when coefficient matrices are restricted vs not - restricted

c. *Graphs*

One of the most general data representation are the graphs, each data representation from before can easily be represented as a graph. The generality of this representation comes with cost of structural properties (e.g. above and bellow being implicitly represented in a grid structure).

i) Expectation Maximization (EM)

Expectation Maximization represents an iterative algorithm that tries to find the Maximum Likelihood Estimate or MLE or Maximum A Posteriori (MAP) for parameters of a method. This method represents a general implementation for any model that provides likelihood on the given data. Guzdial and Riedl managed to use a hierarchical clustering approach by using K-means to train their model. In order to train their model, they used frames from videos with gameplay of *Super Mario Bros.*, model that will be used latter to generate new levels [16]. They parsed each frame from the video with OpenCV. Each frame is later combined in order to form chunks from the level

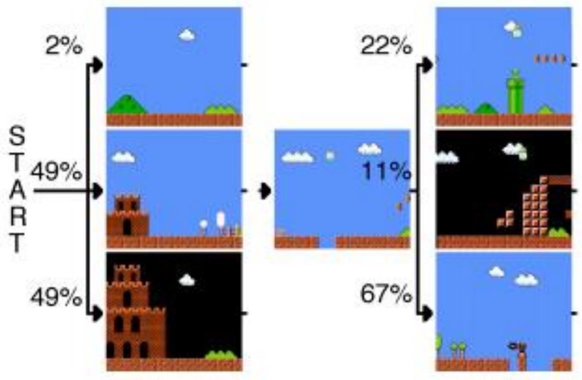


Figure 9: Probability for transitioning from one level chunk to another

geometry which will be later used as input in level generation. They used a graph structure that encoded styles of shapes and the relationship between them with the probabilities. The first step was clustering the chunks of levels in order to derive the style of level chunks, after that we derived the styles of shapes from clustering the shapes within that chunk, and as a last step we determine the relation between shapes by clustering them and finding how they could be combined to generate new levels.

3.Use cases

A. Data Compression

One of the firsts motivations of the PCG in early games but also applicable today in modern games was data compression. Not being enough space on the disk in order to save all the levels and all the maps from a particular game, also some games are represented in vast universes, we needed to generate the surplus of data and deliver it to the player. Games like *No Man's Sky* that contains an almost infinite content space, is impossible to be save, thus the developers used PCG in order to create plants and animals, saving on disk only distinctive features and characteristics. In my opinion autoencoders will be best suited for this kind of work.

B. Co-creative and Mixed Design

Another important factor in game development is represented by design. Procedural content generation using Machine Learning can help the human designer to create diverse content. The "Co-creative" part comes from the fact that the designer is able to train the model and the algorithm by providing examples in the domain. This procedure can reduce dramatically the time dedicated for user training, also it can lower the entry barrier for the designer because it will have much less work to do in comparison with a designer that is working alone. Also, the budget of the project pe be drastically reduced, not having to employ as many designers as before. Also, PCGML algorithms are best suitable to auto-complete game levels and content that is only partially specified by the designer and not yet finished. For example, image inpainting, become popular, a procedure where a NN is able to fill parts of an image that were missing [17].

C. Repair

PCGML are able to identify and repair areas that are not playable or offer a suggestion on how you can fix them. Jain et.al. as presented above is using in his autoencoders a sliding window that is able to repair unplayable game content or illegal segments.

D. Analysis and Critique

Another use case for the PGML that is different from the normal PCG is represented by the capability of recognition, critique and analysis of game content. By creating a model based on the original levels of a specific game, or original content, the model will be later able to identify the content which was created by an algorithm, a designer or a player.

E. Autonomous Generation

Last but not least of the applications of PCGML is represented by the possibility of a continuum creation of content without human interaction. This is not only present in PCGML, autonomy is desired from almost everything in the field of AI. One of the main uses for autonomous generation is the possibility of online content generation for games. It is desirable for a game to be able to generate content as it goes, without having to be adjust by a human, or without having to load the content/level/maps from the disk.

4. Observations

We are able to observe that the platform level generation is the most popular target for generation of content using procedural content generation via machine learning. Also, we can see that the graph is the most general approach of representing the data in PCGML. During the level generation we took into consideration only the playability of the level we generated but there are several more factors that must be taken into consideration like being able to transfer the feedback of the player into the model and also being able to manage the diversity of the level we generated.

5. Conclusions

In this article we gave an overview over the different ways of creating content in games using machine learning algorithms. We observed how Procedural Content Generation is playing a major role nowadays. But, we still have a lot of work to do in order to achieve a more complex level generation. As we can see the main focus right now is on platform games but in the future, we hope that will be possible to generate 3D – levels, items, and characters. There will always be plenty of interesting things to create and discover.

References

[1] N. Shaker, J. Togelius, and M. J. Nelson, Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer, 2016.
[2] A. M. Smith and M. Mateas, “Answer set programming for procedural content generation: A design space approach,” Computational Intelligence and AI in Games, IEEE Transactions on, vol. 3, no. 3, pp. 187–200, 2011.
[3] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey,” Computational Intelligence and AI in Games, IEEE Transactions on, vol. 3, no. 3, pp. 172–186, 2011.
[4] Goodfellow, Ian, et al. *Deep Learning*. MIT Press, 2017.
[5] E. J. Aarseth, Cybertext: Perspectives on ergodic literature. JHU Press, 1997.
[6] S. Dahlskog, J. Togelius, and M. J. Nelson, “Linear levels through n-grams,” in Intl.Academic MindTrek Conf., 2014.
[7] A. K. Hoover, J. Togelius, and G. N. Yannakis, “Composing video game levels with music metaphors through functional scaffolding,” Comp. Creativity & Games at ICCC, 2015.
[8] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis, “Transforming exploratory creativity with delenox,.” in ICCC, 2013, pp. 56–63.
[9] A. Summerville and M. Mateas, “Super Mario as a string: Platformer level generation via LSTMs,” DiGRA/FDG, 2016.
[10] M. Milewicz, RoboRosewater. https://twitter.com/roborosewater?lang=en, 2016.
[11] W. Ching, S. Zhang, and M. Ng, “On multi-dimensional markov chain models,” Pacific Journal of Optimization, vol. 3, no. 2, 2007.
[12] S. Snodgrass and S. Ontañón, “Learning to generate video game maps using Markov models,” TCIAIG, 2016.
[13] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, “Autoencoders for Level Generation, Repair, and Recognition,” in ICCC, 2016.
[14] S. Lee, A. Isaksen, C. Holmgård, and J. Togelius, “Predicting Resource Locations in Game Maps Using Deep Convolutional Neural Networks,” in AIIDE. AAAI, 2016.
[15] N. Shaker and M. Abou-Zleikha, “Alone we can do so little, together we can do so much: A combinatorial approach for generating game content.” AIIDE, vol. 14,2014.
[16] M. Guzdial and M. Riedl, “Game level generation from gameplay videos,” in AIIDE, 2016.
[17] C. Guillemot and O. Le Meur, “Image inpainting: Overview and recent advances,” IEEE signal processing magazine, vol. 31, no. 1, pp. 127–144, 2014.