Generative spaces: assisting the level designer in creating world maps

Abstract

Creating two-dimensional tile maps by hand is a long and tedious process. Tile maps contain an inherent structure and repetition that we should be able to exploit. We provide a way of generating these maps automatically using machine learning techniques. With these set of tools level designers will be able to create worlds that are much larger in scope.

1 Introduction

The development of a video game requires a great variety of assets. A typical game combines music, art and code to form a coherent whole.

This need for resources is one of the reasons why games are so costly to develop. Procedural content generation—the automatic creation of content—has emerged as an answer to this problem and is currently widely used, particularly in studios with few resources. For example, *No Man's Sky (2016)* is a space exploration game that uses procedural generation to generate a whole universe of over 18 quintillion planets $1.8 \cdot 10^{19}$, suffice to say that you would need many lifetimes to explore them all.

Most approaches to procedural generation are rule based, meaning that they rely on the definition of explicit rules. L-systems are an example of such a rule based system. they allow the creation of tree-like structures. TODO: Reference to an image of an L-system. The limitations of rule based systems lies on the fact that they cannot learn from already existing examples and reproduce them. We would not be able to show an L-system images of oaks for example and expect it to reproduce trees in the same style. We would need to define exactly the rules that represent "oakness", the attributes that make an oak an oak. TODO: citation L-systems

Data-driven techniques for procedural generation are still relatively rare; in this paper we try to see whether it is possible to reproduce the spatial style of a given game using data-driven techniques.

- We show how 2d Markov chains can be applied to the generation of 2d tile-maps that are non-linear in nature. As opposed to past litterature Snodgrass and Ontañón (2013) we use maps with a larger number of tile types. Section 2 provides a few background information on what tile-maps are and their current uses while Section 5.1 describes the technical side of their implementation.
- We argue that the generated maps keep the spatial style of the originals by comparing compare how different input maps influence the generated maps in Section 5.2.
- We come up with a backoff smoothing method appropriate for 2d chains. First we explain why it can be useful in Section 3 and then we give the technical details of its implementation in Section 5.3.
- We provide a C# implementation of a text based 2d Markov chain which is accessible at the following link[1].

2 Background

First of all let us define more precisely what a tile-map is. Tile-maps are a staple of many genres of games, they are formed by individual tiles of predefined size and assembled into a grid.

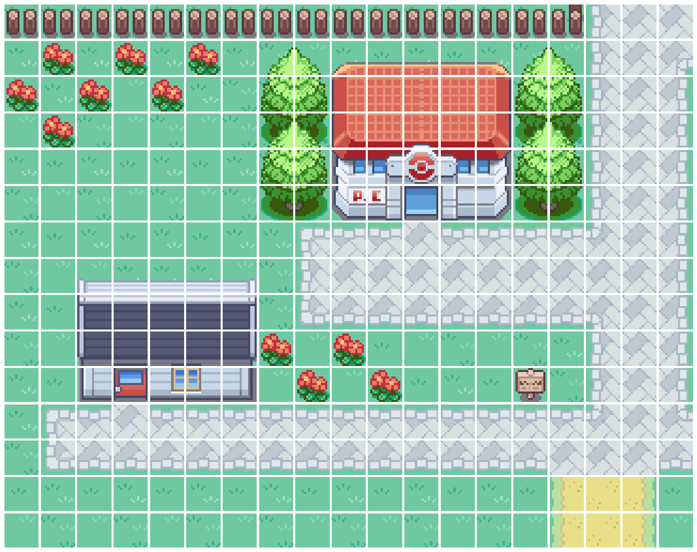

Figure 1: An example of a tile-map with the grid structure visible.
Pokemon FireRed (2004)

[1]https://github.com/stonecauldron/markov2d

TODO: add one figure with multiples tiles inside it

(a) sub-figure 1 caption

(b) sub-figure 2 caption

Figure 2: Caption of figure

Game developers make use of tile-maps for various reasons:

- They reduce the number of art assets that need to be created for the game since the same tile can be reused in multiple places.
- Levels created with tile-maps can be easily represented in a digital format; we can simply represent them as a matrix where each entry identifies a tile uniquely.
- They allow the level designer to create new levels without having to worry about creating new art assets. In some cases this can be extended into fully-fledged level editors where the players themselves can create new levels[2]
- They permit the separation of gameplay and visual appearance. Tiles can for example represent non-traversable space independently from their visual appearance.

Traditionally tile-maps are built manually: the level designer chooses one tile for each cell of the grid. Given that the grids can be very large, we can easily imagine that this is a time-consuming process. It puts a human limit into the size of the maps that can be used in a game. Being able to generate these maps procedurally would lift a burden off the shoulders of the level designers and allow the generation of much larger worlds.

The use of tile-maps might seem like an outdated method but they are still used in many independent games, some of which have been quite successful.

TODO: example of games with tile-maps

3 The idea

Let us remind you of our objective; we want to be able to generate tile-maps from a set of representative examples. Our process has the following steps:

[2]A striking example is the game *Super Mario Maker* which is built entirely around the concept of players creating *Mario Bros.* levels.

1. We choose the set of maps whose spatial style we want to reproduce.
2. We represent the maps as matrices where each entry contains a tile type identifier. This will be our input dataset.
3. We generate a Markov chain whose probability distributions reflect those of the original dataset. This is the learning phase of the model.
4. We then sample this Markov chain to generate a map of any given dimension.

This is comparable to the case where Markov chains are used for automatic essay writing TODO: reference to Murphy book. These automatically written essays use similar idioms and vocabulary to the ones present in the learning dataset; in a certain sense they emulate the style of a given text. For the same reasons we expect our Markov chains to reproduce the spatial style of its learning examples.

One of the problems that appears when sampling Markov chains is that is likely for a chain to generate a sequence of elements that do not exist in the training dataset. As an example take the word predictors that are present in most mobile phones today; Imagine you have just typed the words "Colorless green ideas sleep"[ˆThis sentence is an example of a grammatically correct sentence with no meaning which we can assume does not occur in a standard English text. Taken from Chomsky 1956], since this sentence is not present in the training corpus, the Markov chain will not be able to infer what the next most probable word is and will have to stop generating text.

A way of solving this problem is to use what is called backoff smoothing TODO: Murphy citation in which we consider less elements of the chain when faced with unknown data. To take our above example again, instead of considering the whole chain to infer the next most likely word we can restrict ourselves to the previous word only, which in this case is "sleep". By reducing the number of elements we consider in the chain, we increase the chances of finding equivalent sequences in the training data, thus improving the robustness of the model.

While the definition of backoff smoothing is simple with linear chains, it becomes more complicated in the case of 2d chains. Our approach is detailed in Section 5.3.

4 Results

In this section we will try to evaluate qualitatively a set of maps generated with our method. To this end we will use the data from the game *Pokemon Red (1996)* and *Pokemon FireRed (2004)*. The *FireRed* version is a remake of the *Red* version meaning that is mainly a visual upgrade of the first game; the locations and the spatial distribution of both games are very similar. This allows us to see first hand the effects of an increase in the number of unique tiles: *Pokemon Red* has 125 unique tiles while *Pokemon FireRed* has 1207 different tiles.

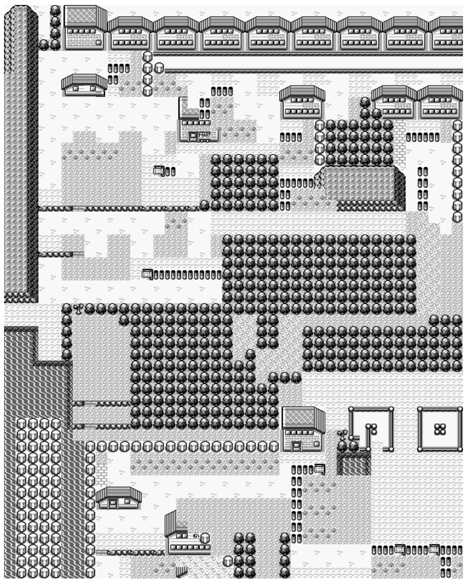

Figure 3: A generated map based on *Pokemon Red*

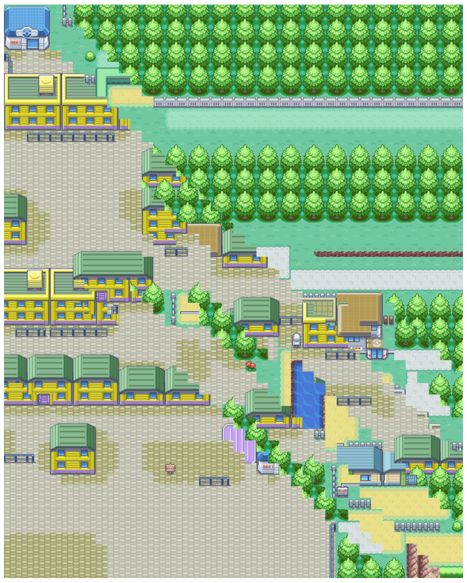

Figure 4: A generated map based on *Pokemon FireRed*

In fig. 3 the overall results looks quite promising:

- Except a patch of grass in the center, almost all the navigable space in the map is reachable.
- Aesthetically the map is plausible; it is comparable to what we can find in the original *Pokemon Red* game with a similar distribution of tiles.
- There are few incomplete structures in the map, mainly the buildings. This is due to the fact that they are the structures with the largest number of dependencies in the tiles that compose them.

In fig. 4 the result is much worse though, there is a much larger number of incomplete structures and while the map looks believable locally, the transition between a forest environment and a city environment in the diagonal is problematic.

This is due to the following facts:

- The *FireRed* version has a higher number of tiles that do not make sense when they are taken individually. The trees in *FireRed* for example are composed of multiple tiles while in *Red* they take up only one tile.
- *FireRed* uses many tiles that have similar meaning but different appearances. The visual look of houses in *FireRed* are very varied while in *Red* they are much more homogeneous.
- The problem on the diagonal suggests that there is not enough data for transitions between different environments. The diagonal shape is due to the ways the backoff smoothing works: in this case it has not enough data to generate the tiles and must rely only on the diagonal predecessors to generate the next tile.

These problems come from the fact that the dependencies between the map elements have a farther reach than what can appropriately be captured by the method. This lead us to believe that to work on games with large tilesets we would need a higher amount of data. Alternatively a few possible improvements to the method are discussed in Section 7.

The computational complexity of our method is linear in relation to the training-set size for training and linear in relation to the map size for generation[3]. This implies we can imagine using the method to generate maps dynamically while the game is running providing players with new levels each time.

[3]This remains true as long as we assume to a take a relatively small number of predecessors into account. See Section 5.3 for more details

5 Technical details

5.1 2D Markov chains

Markov chains *TODO: Markov paper* are a mathematical formalism that allows us to model state transitions of a stochastic process. The particularity of Markov chains is that they only take the current state of the system into account to compute the next state. We can for example model the weather by two states: *sunny* and *rainy* that the probabilities of the weather being either sunny or rainy the next day only depend on the current weather. In our use case, the states correspond to one tile type and each map is generated sequentially by sampling the next state

In our use case we represent a tile-map as a matrix where each entry corresponds to a specific tile. The Markov chain generating the map will have one state per tile type and will fill the matrix of the tile-map sequentially by sampling the next state at each entry.

The chains are trained on the example data with a frequency computation. We used the approach that is detailed in Snodgrass and Ontañón (2013).

This assumption that the next state only depends on the current state is called the *Markov property.* This property can be extended to hold on any number of previous states of the system, in which case we talk about higher order Markov chains. We can even assume that the predecessors are not only linear but that they are taken in multiple dimensions. *TODO: show images of Markov chains.*

The number and position of previous tiles that are taken into account are explicitly defined by the predecessors matrix *TODO: add reference to predecessors matrix image.*

As we can see in #fig:1k-map and #fig:1k-map the choice of the predecessor has a big influence on the outcome of the generation. In #fig:1k-map each tile takes only its immediate left neighbour into account, the chain is linear and we can easily see that there are no relations between each rows. In #fig:3k-map we use a predecessors matrix that is two-dimensional and we can immediately see that the generated map produced much more complex spatial structures as evidenced by the presence of elements composed by a combination of tiles such as houses and buildings.

In general, larger predecessor matrices allow the expression of more complex structures, their use however is limited by the fact that they require more data to train. *TODO: MURPHY reference* Imagine we take a predecessor matrix of the size of the training maps, then the trained Markov chain would only be able to generate copies of its training data since it has learned anything else. We thus lose the potential for variability in the maps generated with larger predecessor matrices.

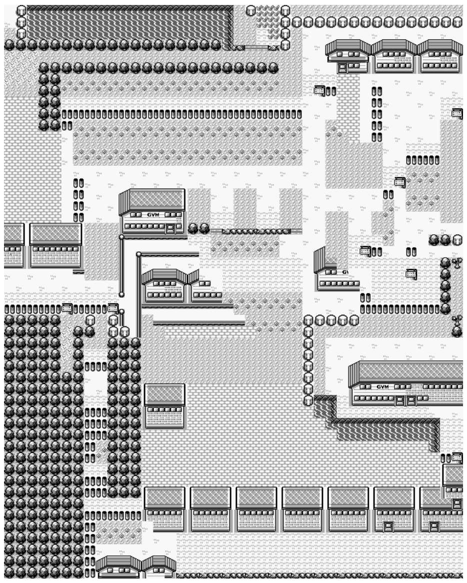
Figure 5: A map generated with a predecessor matrix of size 3

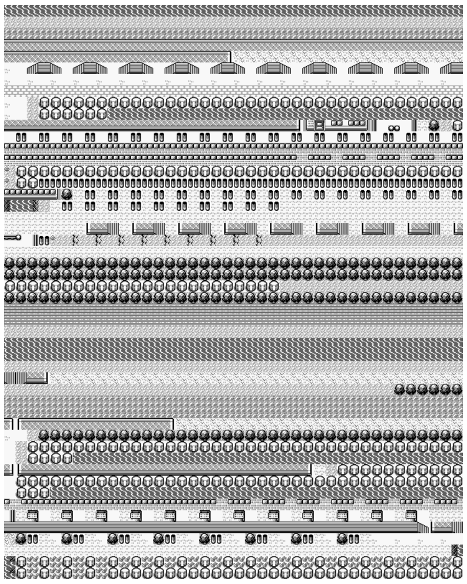
Figure 6: A map generated with a predecessor matrix of size 1

5.2 Spatial style conservation

For our approach to be valid, the generated maps must reproduce the spatial style that is present in the training data-set. If this assumption is true then we expect that if we take two different games as training data that the generated maps will be different not only in the aesthetic sense but also in the spatial sense. As we said in Section 2 we can associate each tile with spatial meaning by defining if it is traversable by the player or not. We can then represent the tile-maps with only two tile types:

- A tile representing empty space depicted as a black tile.
- A tile representing an obstacle depicted as a white tile.

To this end we generate two tile-maps with different games as training data; in fig. 7 we use levels from *Mario* as training data and in fig. 8 we use use training data from *Pokemon*. We can immediately notice differences in the structure of the maps:

- The empty space in the *Mario* map is almost all concentrated in the upper part of the level. This is to give the player room to jump and traverse the obstacles of the level. In the *Pokemon* map however the empty space is much more dispersed; this can be explained by the fact that the player can navigate in two-dimensions.
- We can clearly see a ground-like structure in the bottom of the *Mario* level and a few airborne platforms while the distribution of the obstacles in the *Pokemon* map is much more varied.

The differences in these binary maps can be attributed to the way the players navigate space in their respective games. In *Mario* the player moves through the levels in a left to right direction avoiding enemies and obstacles by jumping; the levels have only one direction of progression. In contrast to the *Mario* game, in *Pokemon* the players can freely move with two degrees of freedom, the progression is non-linear and much less constrained. Given these differences in the generated maps we can assume the Markov chains do succeed in reproducing the spatial style of their training data.

5.3 Backoff smoothing in 2d

As explained before, backoff smoothing is useful when the Markov chain encounters unknown states in the generated data; it allows us take less predecessors into account when needed.

With linear Markov chains the backoff smoothing is easy to define: for example if we take three previous states into account normally, we can take two instead. For 2d chains the problem is not so trivial since it is not clear which elements we have to take and how many of them.

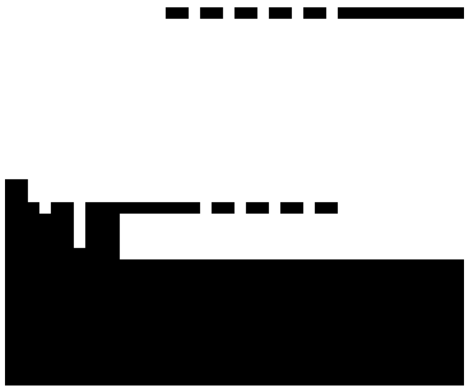
Figure 7: A binary spatial-style map based on *Super Mario Bros.* levels

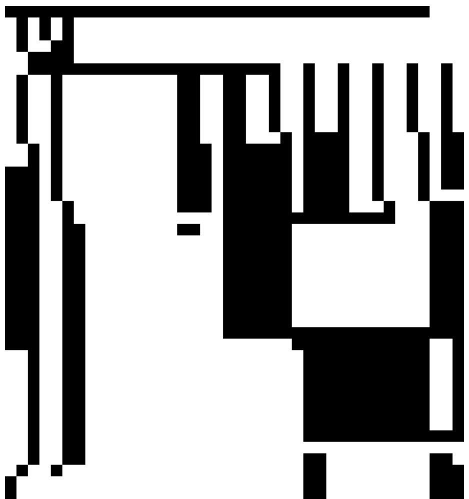
Figure 8: A binary spatial-style map based on *Pokemon Red* maps

To facilitate our discussion we say that a predecessors matrix S is a sub-matrix of predecessors matrix P if it satisfies the following requirements:

- S must take less elements into account than P
- S must not take into account elements that are at a larger distance than P

We can then simply implement backoff smoothing by taking a sub-matrix of our current predecessors matrix. This is done in a recursive manner by removing one element at the largest distance at a time. More formally, the list of sub-matrices of a predecessor matrix P is given by:

$$subMatrices(P) := \begin{cases} \bigcup_{k \in K} subMatrices(k), & \text{if } k \neq 0. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

where K is the set of all matrices that are equal to P minus one of the elements at the largest distance.

$$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{pmatrix}$$

Figure 9: A matrix showing the notions of distance, each number gives the distance of the corresponding cell

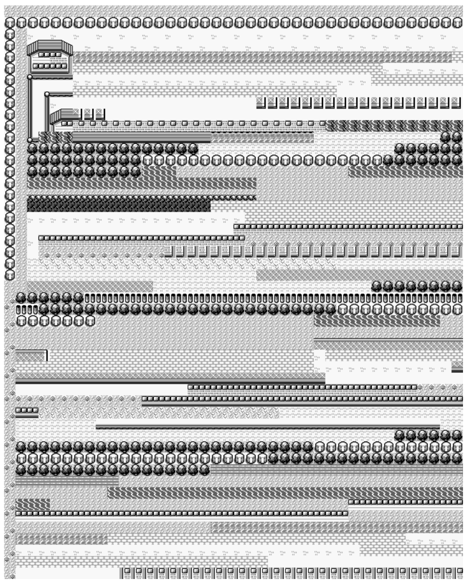

Figure 10: A *Pokemon* map generated without backoff smoothing.

In #fig:backoff we can see the results of a map generated without backoff smoothing. The maps starts off well with the house on the upper left but then quickly degrades as soon it encounters a combination of tiles that are not present in the training data. By mitigating this effect backoff smoothing significantly improves the overall quality of the generated maps. A drawback of the backoff smoothing method is that it increases the computational time of both generation and training exponentially with the size of the predecessors matrix, this implies that in practice it limits the use of very large predecessors matrices.

6 Related work

Game companies generally do not publish their procedural content generation methods so it is hard to evaluate what is common in the industry. In academia, researchers have looked into various approaches for automatic level generation. The game *Super Mario Bros.* in particular has received much attention.

Shaker et al. (2012) uses generative grammars combined with an evolutionary algorithm to create *Mario* levels that adapt themselves to the player's experience. In another insightful paper, Sorenson and Pasquier (2010) also uses an evolutionary algorithm to create a generic framework for level creation. Their approach allows the level designer to enforce multiple constraints on the generated levels making it adaptable to multiple genres of games.

Rather than being evolutionary, our method draws from machine learning techniques. The designers do not have to explicitly articulate rules about their designs but simply need to provide the system with examples of what they want to create.

Papers using machine learning methods to generate content are still a minority, bu the trend seems to be on the rise. Our use of Markov chains is directly inspired by the work of Snodgrass and Ontañón (2013) and Dahlskog, Togelius, and Nelson (2014). They both use Markov chains to create *Mario* levels with very convincing results. The difference with our approach lies in the fact that *Mario* levels have a linear nature. The maps we focus on are non-linear; the player can move through both dimensions. Additionally we use maps with a much higher number of tiles: *Pokemon red* has a total number of 125 unique tiles, *Pokemon FireRed* has 1207 unique tiles, in contrast the maps used by Snodgrass and Ontañón (2013) have only 11 unique tiles. Our sample is more representative of modern games in which the technological constraints limiting the number of unique tiles have almost disappeared.

Adam J Summerville and Mateas (2015) explore non-linear level generation in the context of dungeons for the game *The Legend of Zelda.* Their use of a graph to represent the game space comes from Dormans (2010). Both papers gave us the idea of using a graph to define the high level structure of the world. Our focus is more on world maps rather than dungeons. Dungeons have a clear compartmentalization of individual rooms while world maps have a more organic unfolding of space. Furthermore, dungeons have elements such as locked doors that restrict the order in which the player can traverse the rooms. World maps typically do not have these types of constraints.

Finally, the video game level corpus provided by Adam James Summerville et al. (2016) supplied us with the necessary data to generate *Super Mario Bros* maps.

7 Conclusion and further research

While not perfect, our method shows that Markov chains also have potential for the generation of non-linear maps particularly when the number of unique tiles is relatively small.

There are many ways our method could improved:

- We could create quantitative measures of map quality, generate a large number of them and then select only the best. Since maps can be created relatively fast this should be feasible, the only drawback being that is hard to define measures of map quality that are independent of the game being used.
- We could find ways of grouping semantically similar tiles into the same high-level concepts and thus reduce the total number of unique tiles in the tile-maps.
- Instead of using 2d Markov chains, we could use Markov random fields *TODO: reference MURPHY BOOK*. Their main difference with Markov chains lies in the fact that they encode dependencies between their elements in a non-sequential way. Since we want to create non-linear maps they might be better suited for the task. The disadvantage of using Markov random fields comes from the fact that they are harder to implement and they both require more time to train and to sample appropriately.

Our approach is not restricted to games and we can easily imagine using this method with 3D models composed of voxels. Tiles would be replaced by voxels and the Markov chains would be extended to work in 3d.

In the end though, machine learning techniques for content generation have inherent limits that make the removal of human intervention not conceivable at the moment:

- All the data used for training is created by humans, we cannot expect to use these methods in areas where data is scarce or non-existent.
- Machine learning techniques have a hard time capturing high-level themes, their creations make sense at a local scale but fail to reproduce the holistic feel of the training data. In the case of a game, levels are often organized with the intention of teaching a player a new gameplay mechanic. For example, the underlying theme of a *Mario* level could be teaching the player how to handle flying enemies, all the level would be built around this motif. A data-driven algorithm trained on all the available levels in

> *Mario* would mix data from thematically different levels and thus fail to accurately portray the intention of the designers. In theory this problem could be solved by having a separate dataset for each possible theme in a game, in practice though designers only create one example for each theme which is clearly not sufficient.

In conclusion, we do not see these methods as a replacement for the designer but rather as a tool, taking over in the most repetitive parts of the work. The designer is then free to focus on what really matters: transmitting his creative vision.

References

Dahlskog, Steve, Julian Togelius, and Mark J. Nelson. 2014. "Linear levels through n-grams." *Proceedings of the 18th International Academic MindTrek Conference on Media Business, Management, Content & Services - AcademicMindTrek '14*, 200–206. doi:10.1145/2676467.2676506.

Dormans, Joris. 2010. "Adventures in level design: generating missions and spaces for action adventure games." *… Workshop on Procedural Content Generation in Games*, 1–8. doi:10.1145/1814256.1814257.

Shaker, Noor, Miguel Nicolau, Georgios N. Yannakakis, Julian Togelius, and Michael O'Neill. 2012. "Evolving levels for Super Mario Bros using grammatical evolution." *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012*, no. 1: 304–11. doi:10.1109/CIG.2012.6374170.

Snodgrass, Sam, and Santiago Ontañón. 2013. "Generating Maps Using Markov Chains." *Aiide*, 25–28. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE13/paper/viewPDFInterstitial/7447/7632.

Sorenson, Nathan, and Philippe Pasquier. 2010. "Towards a Generic Framework for Automated Video Game Level Creation Applications of Evolutionary Computation" 6024: 131–40 ST–Towards a Generic Framework for Auto. doi:10.1007/978-3-642-12239-2_14.

Summerville, Adam J, and Michael Mateas. 2015. "The Learning of Zelda : Data-Driven Learning of Level Topology," no. Fdg.

Summerville, Adam James, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. 2016. "The VGLC: The Video Game Level Corpus." *Arxiv*. http://arxiv.org/abs/1606.07487.