Exploring Procedural Image Generation with Markov Wave Function Collapse

Stephen Magrowski

1 Introduction

Procedural Content Generation (PCG) is a design technique used to generate content using a variety of algorithms. PCG techniques are incorporated in use-cases such as map and narrative generation to allow for randomness in the game content. Examples of games that utilize such techniques include *No Man's Sky*, *Dwarf Fortress* [1], and *Caves of Qud* [2]. Some approaches to PCG include noise-based content generation using a noise function such as Perlin noise [3]. Although noise based techniques can be used for PCG, some of the drawbacks of this approach is the reliance on random distributions which are difficult to manipulate [3]. Unlike noise-based algorithms, constraint solving allows for more controllable PCG tools that are content agnostic [4].

Wave Function Collapse (WFC) has been used in games including *Bad North* [2] to generate maps as well as navigation data in the context of the game. It provides benefits over traditional PCG techniques in that WFC generates content based off of rules inferred from contextual information. By comparing against previously observed patterns, WFC can accurately predict viable choices to use when generating outputs using texture synthesis. Texture synthesis generates a large (and often seamlessly tiling) output image with a texture resembling that of a smaller input image [5]. Implementations of WFC can be seen in multiple programming languages including C# [6], Python [7], JavaScript, C++ [8], Rust [9] and others with applications including 2D map generation [6], 3D map generation [8], poetry generation [10], 2D image generation, and more. In this paper, we will be discussing WFC in the context of generating images procedurally and explore some of the existing methodologies used to generate the same. This paper also proposes a WFC implementation using Markov random fields and analyzes the benefits and drawbacks for this approach.

2 Wave Function Collapse

Wave Function Collapse (WFC) is a constraint solving algorithm named after the quantum physics wave superposition concept. Maxim Gumin first implemented WFC using C# to incorporate a texture synthesis constraint solving PCG algorithm [6]. Entropy in WFC is described as the set of all possible color outcomes allowed by the input constraints for each individual pixel to be generated in the output image. This means that if a set of n colors were observed in an image, the entropy for any pixel in that image can be up to n. By following a minimum entropy heuristic, WFC resolves constraints from a point on the output sample and propagates outwards resolving entropy to create a final pattern [4]. These constraints determine the occurrence of specific information in the input data which can be visualized as neighboring pixel constraints in the case of 2D images. By splitting an image into uniform chunks called kernels, the WFC algorithm can identify patterns that exist in the input sample and their rate of occurrence. Using this information, WFC performs texture synthesis to generate a new output image that follows the same constraints observed in the input. Figure 1 illustrates some of the inputs and their corresponding procedural outputs generated by Maxim Gumin's original implementation of WFC [6].

This paper discusses 2 existing models used to generate procedural content using WFC, they are the Overlapping WFC (OWFC) and Tiling WFC (TWFC) models, and proposes a new Markov WFC (MkWFC) model. In the case of 2D, the OWFC model splits an input image into smaller $N \times N$ sized kernels. These kernels are then overlapped to determine the possible occurrences of pixels in the output. The TWFC model, on the other hand, does not identify the kernels by splitting the input image into chunks, instead, it receives these kernels and their constraints in the form of meta-data. This meta-data can be used by TWFC to propagate constraints and solve the entropies of kernels without having to determine them in a pre-compute step. By eliminating the iterative kernel identification process, the TWFC model generates procedural outputs faster than the OWFC model. This, however, comes at the cost of manually generating a meta-data file that feeds the relevant constraint information to TWFC which is non-trivial; this affects the scalability of TWFC as increasing the complexity of the input can result in the need for increased investment of resources in determining the constraints required to generate desired results [11].

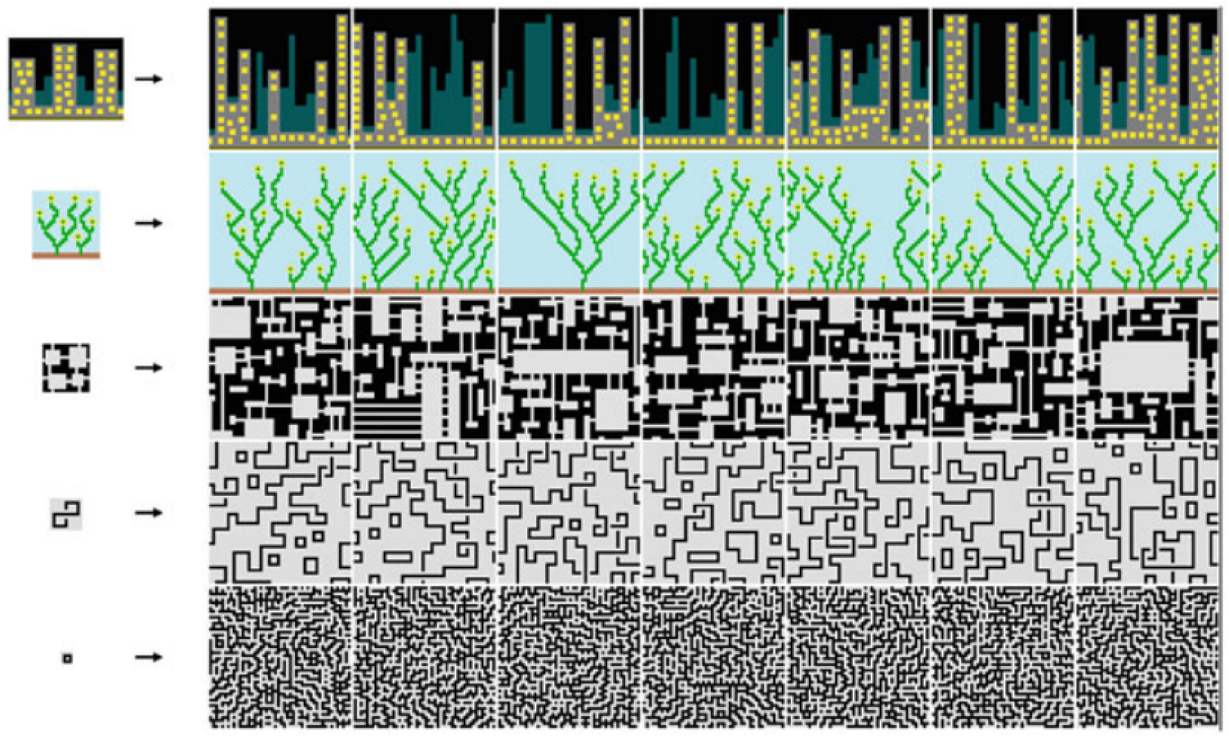
Fig. 1 Samples from Maxim Gumin's WFC repository [6] highlighting the input provided to the algorithm on the left side and the random procedural results on the right side for each corresponding input image

The MkWFC model builds on the TWFC model by using constraint identification from input samples similar to OWFC while maintaining the performance benefits of TWFC.

2.1 *Overlapping Wave Function Collapse Model*

2.1.1 Overview

The OWFC model expects only an input sample as opposed to the other models which require some meta-data or a combination of meta-data and input samples. The OWFC model splits the input sample image into N × N kernels where N is defined by the user; these kernels are then stored in a list to be used in the generation of the output image. These identified kernels account for 8 possible occurrences of each kernel including rotated and reflected versions of the kernel. The requested output image is allocated in memory but none of the pixels in the output image is assigned any color information as all the pixels in the requested output can be any of the pixel colors read from the input. This means they all start at the highest possible entropy (all of the observed colors from the input) and the entropy of the image needs to be resolved to finally generate an image where there is only 1 possible color for any given pixel in the output (i.e., entropy of 1).

To perform the entropy resolution, an arbitrary pixel is selected in the output and assigned one of its possible color values. This reduces the entropy of that pixel to 1 and the assigned color information is then propagated to its neighboring pixels. Each of the neighboring pixels is compared against observed kernels to determine possible color values based on the previously resolved pixel information. The check is performed by overlapping the kernel with the neighboring pixel to determine if the entropy for the neighboring pixel is valid based on the occurrence of color values in the kernel. If the entropy values exist in the overlapped kernel, the pixel entropy is maintained, otherwise, all entropy values that cannot occur are eliminated from the neighboring pixel entropy [12]. This results in the entropy of the neighboring pixel either decreases or remains the same. The kernel overlap is performed for all neighboring pixels; the process continues recursively until all the pixels in the output image have been resolved to an entropy of 1. Due to the nature of the constraints used by the algorithm, there are instances where entropy resolution cannot resolve the output pixel entropies [13]. This occurs when there is not enough information to identify constraints that can resolve the pixel entropy, in which case WFC cannot generate a result with the provided input [14].

By arbitrarily picking the first pixel to perform entropy resolution, the resolution occurs differently on every execution of the algorithm hence generating different outputs.

2.1.1 Tradeoffs of OWFC

Although OWFC does provide procedural results, due to the iterative nature of kernel identification OWFC is computationally heavy. By increasing the size of the kernel, there is a rapid increase in computational time in the order of N2 where N*N represents the dimensions of the kernel. Increasing the complexity of the image by using numerous colors or well-defined patterns such as alphabets or symbols causes results to lose randomness due to there being a larger constraint set. While providing procedural results, OWFC suffers from high computational cost as well as the lack of scalability.

2.2 *Tile-Based Wave Function Collapse Model*

2.2.1 Overview

The TWFC model is very similar to the OWFC model except for the kernel generation step. Instead of computing the kernels present in an input sample, the TWFC model relies on a meta-data file that can provide the constraints required to generate an output image. By doing so, the tiling model can benefit from not having to use per-pixel entropies; instead, a tile can comprise a set of pixels that fit on a fixed tile grid. The constraints are neighborhood relationship information between 2 distinct tiles and their orientations. The algorithm uses a WFC wave object which contains all possible tile instances that can occur on the output image. Similar to OWFC, TWFC still requires the need to resolve the entropies of each tile in the output image as the WFC wave object considers all tiles to have the highest possible entropies (i.e., the total number of tiles used by the algorithm) when generating the output [15].

When comparing the 2 models, the OWFC model takes a sample image as input while the TWFC model takes a meta-data file as input. The meta-data file contains a set of tiles to be used as subsets of the generated image and some constraints that determine which tiles can be placed next to each other. By feeding the constraints and possible tiles in manually generated meta-data to TWFC, the kernel generation step is not required hence reducing the computational load involved of TWFC as opposed to that of OWFC. Since TWFC uses tiles on a fixed tile grid as opposed to pixels in an image, the user has control over the size of the tiles used as well as the complexity of the output generated as tiles can be simplified or made more detailed based on usage. Unlike OWFC however, TWFC has no way of identifying the rate of occurrence for kernels (in this case tiles) occurring in the input as there is no input image. Instead, to account for a rate of occurrence, TWFC requires this information to be provided as tile frequency or tile weight in the meta-data file.

2.2.2 Architecture

The architecture used by TWFC is illustrated in Fig. 2. The meta-data is parsed to identify the set of tiles to be used for texture synthesis and their symmetries are computed. Once all tiles have been identified and their symmetries computed, TWFC parses the constraints to be used to solve against. TWFC then uses a WFC Propagator object which will propagate the entropy resolution information and the WFC wave which stores an output of the desired size. After creating the propagator and wave, TWFC can begin the entropy resolution process.

Similar to the case of OWFC, the iterative entropy resolution process cannot complete if there is not enough constraint information, in which case we set a limit to the number of attempts to complete entropy resolution. After the entropy resolution solves for an output WFC wave object, the algorithm then writes the content of the WFC wave object as an output image of the same size. The result of the TWFC algorithm returns either a pass or fail output that conveys if the model generated an output file or not from the given input sample.

2.2.3 Meta-Data Configuration

TWFC uses meta-data comprising tile information utilized by the texture synthesis step along with constraint information. The meta-data lists a set of tiles to be used by TWFC as well as their size in pixels and their possible symmetries (rotated and reflected instances) of the tiles to be used when generating an output. The constraints in meta-data specify a left tile and its orientation to a corresponding right tile and its orientation to determine which of these constraints can fit in the output WFC wave object.

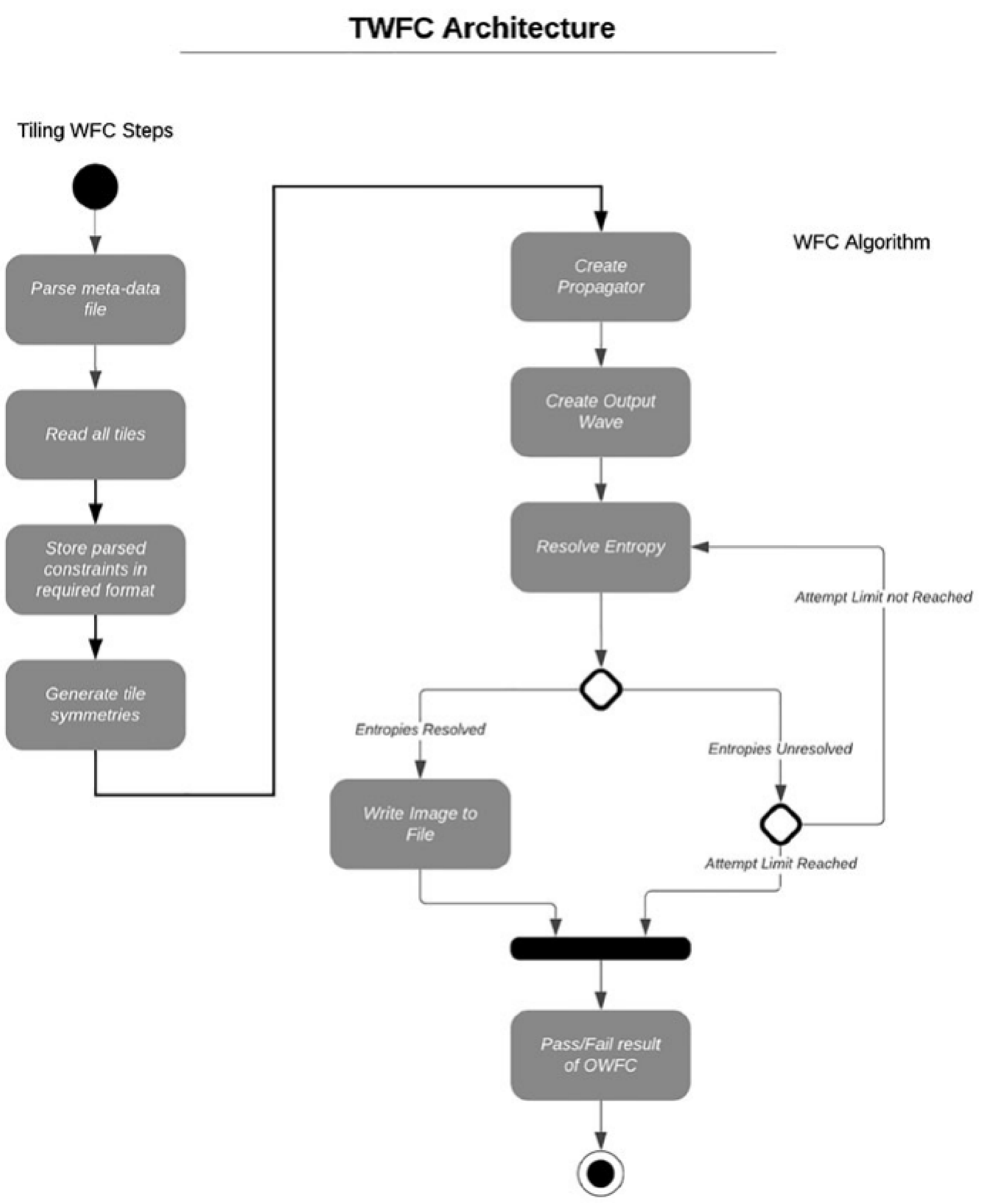

Fig. 2 Flowchart of tiling WFC

2.2.4 Improvements Over OWFC

By eliminating the kernel generation process, TWFC does not need to pre-compute the existing patterns and their rate of occurrence, instead, the meta-data file provides the required tile information including frequencies, constraints, and sizes hence allowing the TWFC algorithm to only perform the WFC wave propagation and entropy resolution. With kernels relationships determined ahead of time, the computational overhead is reduced while also providing users the opportunity to control the constraints involved as well as the complexity of the tiles without directly affecting the WFC resolution and propagation. By doing so, making changes to tiles being used by the TWFC model does not affect its speed, adding constraints however would affect the performance of the algorithm due to the need to do fewer (if neighbor constraints are reduced) or more (if neighbor constraints are increased) comparisons during entropy resolution.

2.2.5 Tradeoffs of TWFC

Despite having a reduced computationally load and improving on the speed of generating output using WFC, the TWFC model still has some drawbacks. The generation of the meta-data file is non-trivial. To identify the required constraints and how they affect the generated output requires designing possible outcomes by involving some constraints as opposed to others or by balancing the frequency of the tiles being used. This can be challenging as the reduction or inclusion of even a single constraint can change the entropy resolution process significantly. Due to the determination of neighborhood constraints being a non-trivial process, TWFC also does not scale well. Increasing the number of constraints or adding more tiles to the TWFC problem requires re-iterating on the possible outcomes and manipulating the non-trivial neighborhood constraint set. As the number of tiles and constraints increases, the complexity also increases which can result in a greater chance of human error. Similar to OWFC, there are cases where TWFC cannot resolve the tile entropy and the algorithm fails to generate an output image. This can be a result of conflicting constraints or human error in curating the manually generated meta-data file.

2.3 *Markov Chain Wave Function Collapse Model*

2.3.1 Overview

The proposed Markov WFC (MkWFC) model aims to solve the scalability issues of TWFC while maintaining the performance benefits it provides over OWFC. To eliminate this scalability tradeoff, MkWFC performs an automation step to infer the constraints rather than use a manually generated meta-data file. MkWFC uses the tile information to scan input images provided in meta-data to determine constraints. These constraints are used to create Markov random fields that determine neighboring tile relationships for each instance of a tile in the input image. Although this introduces an overhead in terms of execution time, by automating the nontrivial step of constraint identification MkWFC identifies constraints and generates a Markov random field to be used by the WFC algorithm. Due to this automated pre-processing step MkWFC can scale better than TWFC with increased input and output sizes.

There are 2 benefits MkWFC provides over the TWFC implementation. The first being the elimination of manually determining non-trivial constraint information. By eliminating this process MkWFC uses the design philosophy of the OWFC model, which is to observe input images for constraint identification. With tile information present in meta-data, the MkWFC model can pre-compute their symmetries and observe the input images for occurrences of these tiles, thereby determining its neighborhood relationships to build constraints that can be used for output generation. The second benefit is scalability in terms of increasing the number of constraints used by the WFC algorithm. Due to the automation of constraint identification, increasing the constraint set used by WFC requires adding additional input images or increasing the size of provided input images to identify a constraint set with greater coverage over the theoretical maximum of possible constraints. This also allows for generating results with different constraints by switching between images that follow the desired constraints as opposed to manually curating new constraint relationships to be passed as meta-data like in the case of TWFC.

After parsing the meta-data, MkWFC generates all the tile symmetries as is in the case with TWFC. After generating the symmetries, MkWFC performs an extra step of analyzing all the input samples provided in the meta-data for tile occurrences. By analyzing the entire image, MkWFC identifies a set of neighborhood relationships between 2 adjacent tiles. For each tile, the MkWFC model creates a Markov random field of compatible neighbors. After identifying compatible neighbors for all tile occurrences in the input images, these neighborhood constraints are propagated during entropy resolution.

2.3.2 Architecture

The architecture used by MkWFC is illustrated in Fig. 3. The meta-data is parsed to identify the set of tiles to be used for texture synthesis and their symmetries are computed as in the case of TWFC. The difference is in the identification of constraints. MkWFC uses input images described in meta-data to determine tile instances and create constraint chains. The constraint chains are then populated into a constraint set of a Markov random field which are the relationships of each tile instance in the input image to its corresponding neighbors. MkWFC then creates the WFC Propagator object which will propagate the entropy resolution information and the WFC wave which stores an output of the desired size. The result of the MkWFC algorithm returns either a pass or fail output that conveys if the model generated an output file or not from the given input sample similar to TWFC.

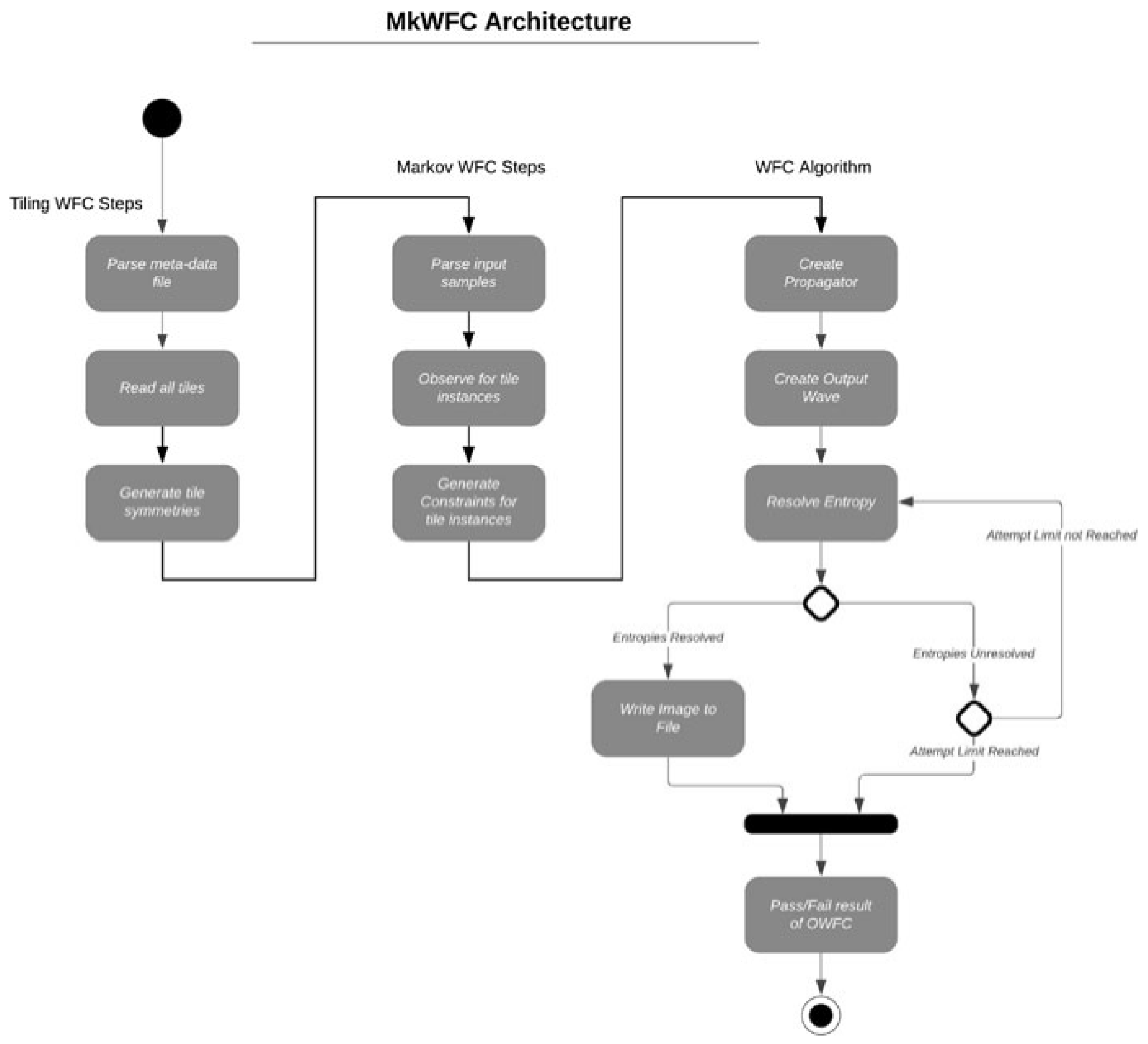

Fig. 3 Flowchart used by Markov wave function collapse model

2.3.3 Meta-Data Configuration

Similar to TWFC, MkWFC also needs some meta-data to describe the tiles used by MkWFC as well as their respective symmetries. So as in the TWFC case, there is a list of tiles to be used provided in meta-data along with their size in pixels and their respective symmetries. But unlike TWFC, instead of a list of left tile to right tile neighborhood constraint information, the meta-data for MkWFC provides a list of input images. These input images are analyzed in the constraint identification step and a list of left tile to right tile neighborhood constraints is automated, then fed as a constraint list to the tiling step of MkWFC. The elimination of curating non-trivial constraint information reduces the size of the meta-data file as well as eliminates the time intensiveness of manually curating the constraint information.

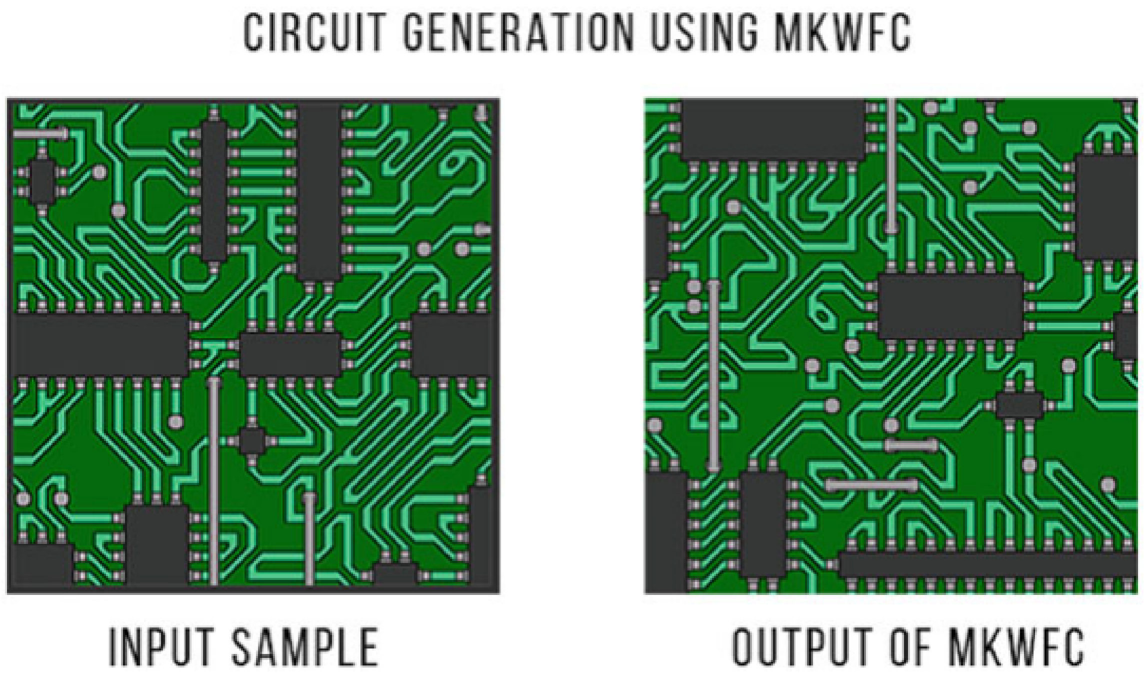

Fig. 4 Input sample and generated output for circuit generation using MkWFC

2.3.4 Model Usage

To showcase the workings of MkWFC we will consider the use-case of procedurally generating a circuit board as described in Fig. 4. This image is passed as input to MkWFC. Its corresponding meta-data file contains a list of 14 tiles that are used and their corresponding symmetry.

As expected, the meta-data only contains tile information and the input samples that would be used by MkWFC which in this case is the image in Fig. 4. On execution of MkWFC, the automated constraint identification step checks against the input image to identify tile occurrences and creates a list of neighborhood constraints as a Markov random field. These constraints are then used by the WFC algorithm to perform the texture synthesis step to resolve the entropy of each tile instance and generate an output image.

On successful entropy resolution, MkWFC saves the resolved output as an image. The results of this problem are illustrated in Fig. 4 where the constraints identified from the input image generated an output image following the same constraints. It is important to note however that similar to OWFC and TWFC, MkWFC cannot guarantee the resolution of entropy per tile as there may be conflicting constraints which may require more than 1 pass to resolve said constraints or the constraint contradictions are impossible to resolve using the identified constraint set.

3 Experiment

To compare the scalability and performance of MkWFC against TWFC, 6 different content generation problems namely Circles, Circuits, Rooms, Knots, Castles, and Summer were considered. Each of the 6 problems was executed to generate outputs of 3 different sizes which were 5 × 5 tiled for small outputs, 10 × 10 tiled for medium outputs, and 25 × 25 tiled for large outputs. Metrics for comparing TWFC against MkWFC were calculated by executing 10 trials for each of the 6 problems when generating outputs for each of the 3 output sizes. A total of 180 trials were performed for TWFC (60 trials each of 5 × 5, 10 × 10, and 25 × 25 tiled outputs). Similarly 540 trials were performed for MkWFC, where the first 180 trials used 5 × 5 tiled input samples to generate outputs of all sizes (60 trials of 5 × 5, 10 × 10, and 25 × 25 outputs), the next 180 trials used 10 × 10 tiled inputs to generate outputs of all sizes, and the final 180 trials used 25 × 25 tiled inputs to generate outputs of all sizes. By averaging the metrics for all 720 trials, this paper compares the constraint identification, usage, and execution times in milliseconds for TWFC and MkWFC.

To generate outputs, TWFC used a manually generated meta-data file for each of the 6 problems which listed the tiles used for that problem as well as the constraints to be solved against. MkWFC on the other hand used 3 different metadata files, 1 for each desired output size, and used 3 images as input samples from which to determine constraints. When generating outputs, MkWFC performance was measured when using 3 input samples of size 5 × 5, 10 × 10, and 25 × 25 to generate outputs of all sizes. All constraints were used as a set of left to right neighboring tile relationships. In the case of TWFC, these were provided in the manually generated meta-data file whereas in the case of MkWFC, they were identified from the input sample images provided to the algorithm. To calculate the total number of theoretical constraints, it was assumed that all tiles used by MkWFC and TWFC had 8 possible symmetries. The formula used to compute the total number of possible constraints on an $n * n$ sized grid is described in Eq. 1 such that we compute 4 possible constraints for all tiles in the n*n image excluding the boundary tiles where n represents the width and height of the image.

$$\epsilon = (\alpha + \beta + \gamma) \tag{1}$$

where α, β, and γ are described in Eqs. 2, 3, and 4 as

$$\alpha = (n - 2)^{\wedge}2 * 4 \tag{2}$$

$$\beta = (n - 2) * 4 * 3 \tag{3}$$

$$\gamma = 4 * 2 \tag{4}$$

ϵ in Eq. 1 represents the maximum possible constraint pairs on a tile grid of size $n * n$. α determines the maximum possible constraint pairs of all the tiles excluding the boundary tiles on the tile grid. The number of tiles on the grid excluding the boundary tiles is represented as (n − 2)2 and since these tiles have neighbors on all 4 directions *top*, *bottom*, *left*, and *right* we multiply 4 as the possible constraint pairs for these tiles.

β represents the maximum possible constraint pairs for the tiles that lie on the boundary of the tile grid except for the corner tiles. The tiles on the boundary except for corner tiles can each have 3 constraint pairs associated with them as one of their 4 directions will have no neighboring tile. This is represented as (n − 2) tiles for each of the 4 sides having 3 possible neighborhood constraints.

Finally a constant γ represents the maximum possible constraints for the 4 corner tiles on the tile grid. Since the corner tiles have neighbors in 2 out of the 4 possible directions, the value of C is $4 * 2$ and is constant.

4 Results

Figure 6 showcases 1 of the input samples used by MkWFC and the outputs generated by both TWFC and MkWFC for all 6 problems in 1 of the trials performed. For the *Circuits* problem, MkWFC identified insufficient unique constraints resulting in fail conditions when generating outputs under 2 of the 3 test conditions. MkWFC was unable to resolve constraints when using small and medium input samples to identify constraints but was able to generate results using large input samples. This lack of information for entropy resolution resulted in entropy for tiles in the output to be higher than 1 which meant that there is more than 1 possible outcome for that tile in the output resulting in a fail condition.

Failure to generate outputs for the *Circuits* problem occurred when the number of constraints identified to the possible constraint pairs on the output image fell below a threshold value. The threshold value identified was 0.2 when using small input samples and 0.47 when using medium input samples. The tiles used to generate outputs for the Circuits problem are shown in Fig. 5 (Fig. 6).

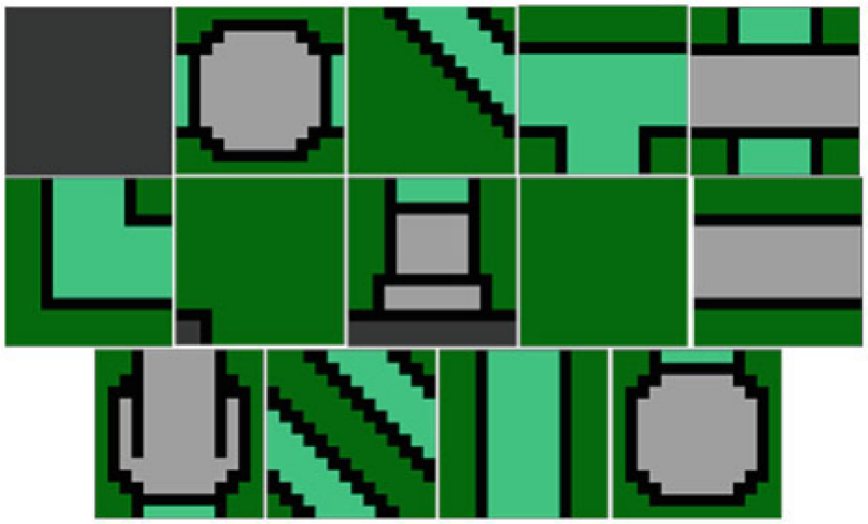

Fig. 5 Tiles provided as inputs for TWFC and MkWFC for the *Circuits* problem

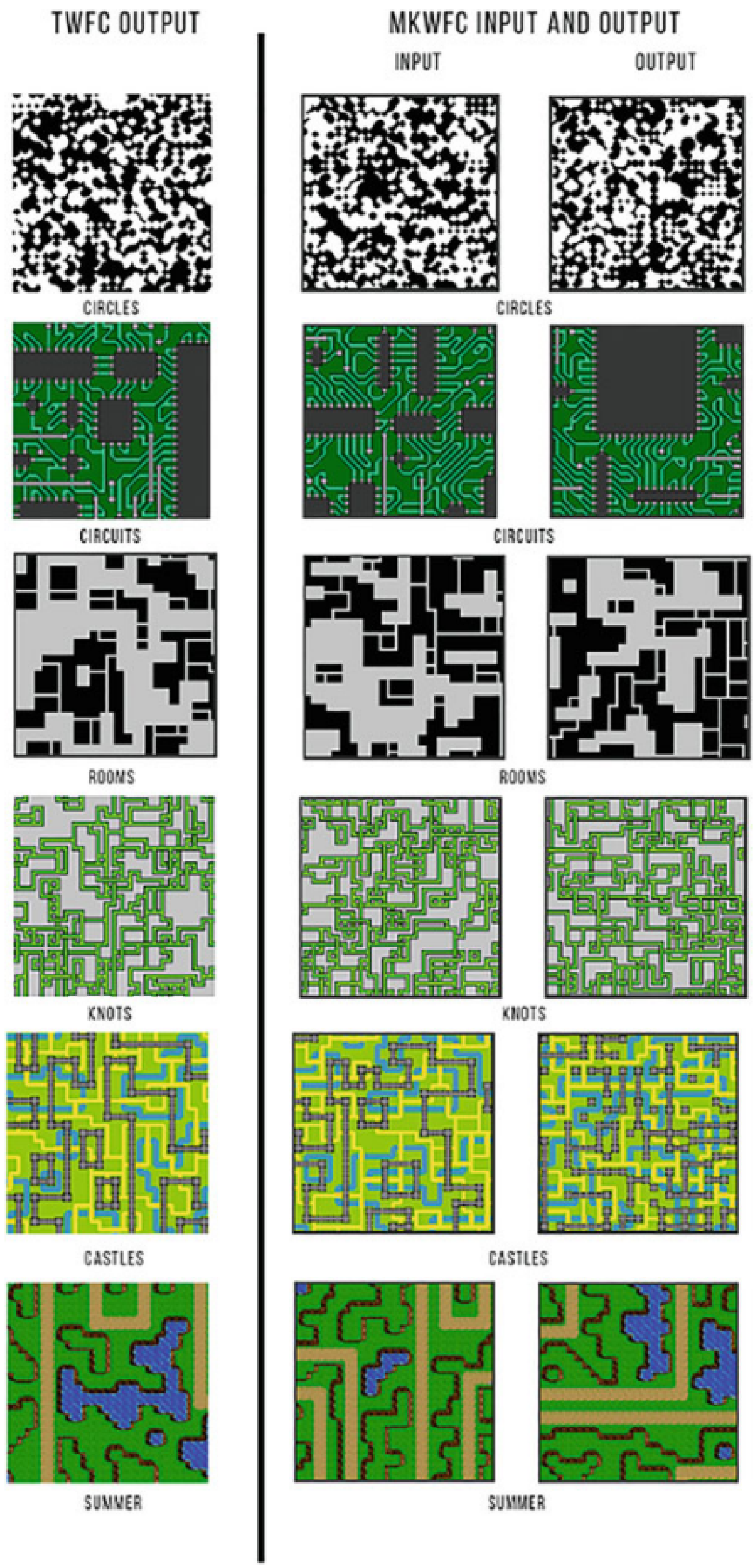

Fig. 6 Outputs of TWFC and input sample with corresponding generated outputs using MkWFC for 1 of the trials

The TWFC model relies on the use of constraints to be provided in meta-data which would require the need to explicitly specify the desired constraints to use whereas MkWFC identifies all constraints existing in an input sample and generates Markov random fields of all the neighborhood tile constraints. As a result, there are constraints identified by MkWFC which may not be part of the constraint set provided to TWFC resulting in patterns in output which would not be generated by the TWFC implementation unless explicitly mentioned in manually generated metadata. To highlight the case where new constraints are used in generating outputs by MkWFC, the *Castles* trial used by MkWFC is showcased in Fig. 7 where the tile instances highlighted in red represent the constraints that were identified by MkWFC which were not present in the TWFC implementation. When comparing TWFC and MkWFC, there was an increase in the coverage of the maximum theoretical constraints set for MkWFC when compared to TWFC based on the data analysis performed.

Generating the constraints in meta-data for TWFC requires the use of reference images provided by a designer, by automating constraint identification MkWFC used the same reference images to identify constraints in millisecond range. MkWFC consistently increased the number of constraints identified as a result of automating the constraint identification process and scaled with increasing output sizes. MkWFC introduces an increase in execution time which is nominal to allow for performance in real-time similar to TWFC.

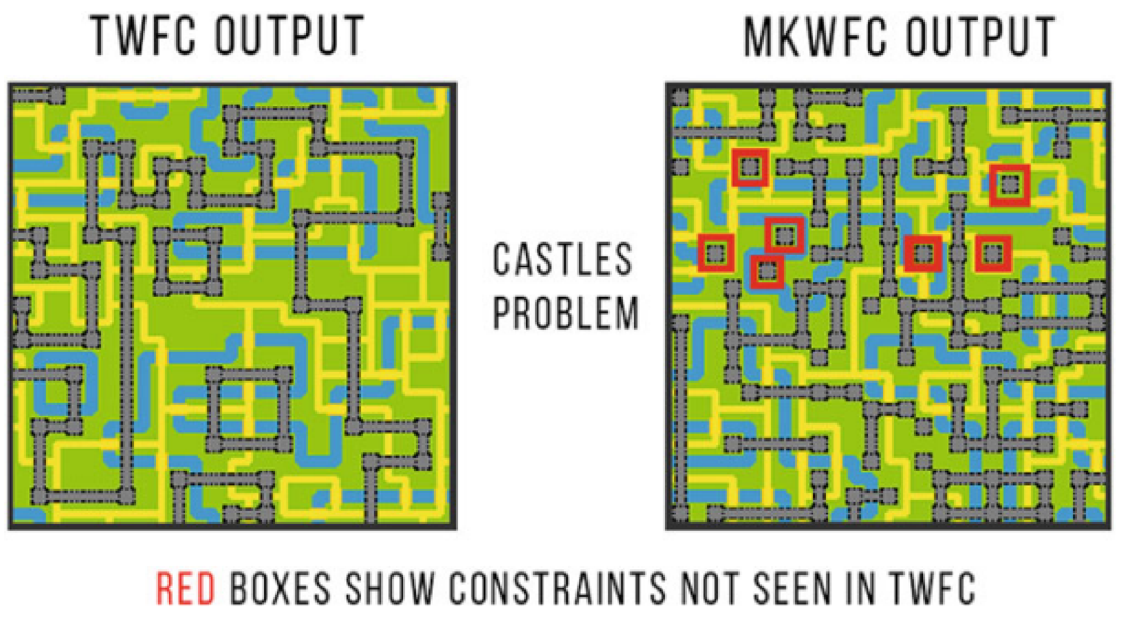

Fig. 7 Comparison of TWFC and MkWFC outputs for the *Castle* problem. Red boxes show new constraints not present in TWFC

4.1 *Constraint Identification*

Table 1 is the comparison of constraints received by TWFC to those identified by MkWFC when generating outputs using 3 image samples as inputs. When reading constraints, TWFC received 88.76 ± 61.16%(α = 0.05) of the constraints set identified by MkWFC when using small image samples of size 5 × 5. Each identified constraint in MkWFC was weighted to determine the frequency of a pattern that existed in input. As the size of input images increased, the constraints received by TWFC reduced to 46.95 ± 19.77%(α = 0.05) of the constraint set identified by MkWFC when using 10 × 10 sized input samples and further reduced to 36.47 ± 6.2%(α = 0.05) of the constraint set identified by MkWFC when using 25 × 25 sized input samples.

From Table 1 it is clear that MkWFC manages to identify more constraints when compared to those read by TWFC with increasing input sizes. As the size of input sample increases, the constraint set identified by MkWFC also increases whereas the TWFC model utilizes the same fixed constraint set which needs to be manually generated for each problem

4.2 *Constraint Usage*

Table 2 is the comparison of constraint usage for TWFC and MkWFC when generating outputs of varying sizes. The table compares the outputs generated when using 3 different sized inputs with MkWFC. When generating outputs using inputs of size 5 × 5, TWFC used 42.65 ± 5.72%(α = 0.05) of the constraints used by MkWFC as the worst case for TWFC. The best case for TWFC resulted in the usage of 57.38 ± 14.29%(α = 0.05) of the constraints used by MkWFC when generating outputs of size 25 × 25 with 5 × 5 input samples. In the case of using 5 × 5 input samples, MkWFC provides almost double the coverage of constraints used when compared to TWFC. When comparing the constraint usage of TWFC with MkWFC while using 10 × 10 images to generate outputs with MkWFC, TWFC exhibits a best case of 98.72 ± 152.9%(α = 0.05) of the constraint set used by MkWFC when generating 5 × 5 sized output images. This drops to a 37.31 ± 7.12%(α = 0.05) of the constraints used in the worst case of generating 10 × 10 image outputs with the metrics for generating 25 × 25 outputs beings similar where TWFC uses 38 ± 5.4%(α = 0.05) of the constraint set used by MkWFC. The last set of metrics to be compared is when MkWFC uses 25 × 25 sized input samples to generate outputs. The constraint coverage of TWFC exhibits the best case of 67.45 ± 82.12%(α = 0.05) of the coverage used by MkWFC when generating outputs of size 10 × 10. This drops to the worst case of TWFC using 34.57 ± 4.68%(α = 0.05) of the constraints used by MkWFC when generating 25 × 25 sized output images.

Table 1 Constraint identification TWFC compared to MkWFC

Input size	Mean ($\alpha = 0.05$)
5 × 5	88.76 ± 61.16%
10 × 10	46.95 ± 19.77%
25 × 25	36.47 ± 6.2%

From Table 2, MkWFC consistently manages to use a larger constraint converge to generate outputs irrespective of size when compared to TWFC. The optimal case for MkWFC is when the size of generated output is the same as the size of the input samples provided to the algorithm while the worst case is observed when generating outputs that are smaller than the size of the provided input samples.

Table 2 Constraint usage TWFC compared to MkWFC

Input size	Output size	Mean ($\alpha = 0.05$)
5 × 5	5 × 5	42.65 ± 5.72%
5 × 5	10 × 10	45.47 ± 13.17%
5 × 5	25 × 25	57.38 ± 14.29%
10 × 10	5 × 5	98.72 ± 152.9%
10 × 10	10 × 10	37.31 ± 7.12%
10 × 10	25 × 25	38 ± 5.4%
25 × 25	5 × 5	64.83 ± 67.33%
25 × 25	10 × 10	67.45 ± 82.12%
25 × 25	25 × 25	34.57 ± 4.68%

Table 3 Execution time comparisons TWFC and MkWFC

Input size	MkWFC constraint identification time ($\alpha = 0.05$)	MkWFC tiling step time ($\alpha = 0.05$)	TWFC execution time ($\alpha = 0.05$)
5 × 5	1.03 ± 0.09 ms	22.76 ± 2.94 ms	25.12 ± 4.77 ms
10 × 10	4.13 ± 0.38 ms	34.83 ± 6.03 ms	25.54 ± 5 ms
25 × 25	28.19 ± 2.58 ms	60.71 ± 8.82 ms	25.15 ± 4.6 ms

4.3 *Performance*

Table 3 shows the time taken on average by MkWFC for constraint identification and tiling step when compared to the total execution time of TWFC.

When generating outputs using images of size 5 × 5, MkWFC spent 1.03 ± 0.09 ms on constraint identification and 22.76±2.94 ms on the tiling step. This result shows there was no statistical significant difference between the 25.12 ± 4.77 ms TWFC execution times at this size. MkWFC spent 28.19 ± 2.58 ms on constraint identification and 60.71 ± 8.82 ms on the tiling step for 25 × 25 tiled steps, which was a statistically significant increase from the 25.15 ± 4.60 ms execution time of TWFC. Although TWFC performed faster for the same case, MkWFC identified almost 3 times the number of constraints manually identified for TWFC. The increase in constraints identified introduces an increase in execution time in the range of milliseconds. To provide the same increase in constraints for TWFC, the manual meta-data generation process would require a substantial increase in design time. Due to the increasing number of constraints identified, the tiling step of MkWFC exhibits a slower execution time with increasing image sizes.

5 Conclusion

MkWFC managed to consistently provide larger constraint coverage for both the constraint identification and usage when compared to TWFC for all 3 image sizes. The best case usage of MkWFC was exhibited when generating outputs of the same size as the input samples provided to the algorithm. Due to an increase in constraints used by MkWFC, the tiling step time was larger than the total execution time of TWFC when generating outputs of increasing size. The increase in execution time of MkWFC is in the range of milliseconds maintaining the real-time performance advantages of TWFC.

Where run-time performance is a priority, TWFC can be used with a substantial design time to generate meta-data to save on the run-time time overhead of MkWFC. MkWFC however, consistently provides increased constraint usage across all image sizes making it better suited for increasing content sizes. The added performance overhead in automated constraint identification is significantly less than the design time required to manually generate the constraint set. MkWFC provides the performance benefits of TWFC over OWFC while maintaining viable usage in real-time applications.

References

1. Procedural Generation with Wave Function Collapse (2019). https://gridbugs.org/wavefunction-collapse/
2. J. Grinblat, C.B. Bucklew, Subverting historical cause & effect: generation of mythic biographies in Caves of Qud, in Proceedings of the International Conference on the Foundations of Digital Games - FDG '17 (ACM Press, Hyannis, Massachusetts, 2017), pp. 1–7. http://dl.acm.org/citation.cfm?doid=3102071.3110574
3. Math for Game Developers, End-to-End Procedural Generation in 'Caves of Qud' (2019). https://www.gdcvault.com/play/1026313/Math-for-Game-Developers-End
4. Isaackarth.com, WaveFunctionCollapse is Constraint Solving in the Wild (2017). https://isaackarth.com/papers/wfc_is_constraint_solving_in_the_wild/
5. A. Efros, T. Leung, Texture synthesis by non-parametric sampling, in Proceedings of theSeventh IEEE International Conference on Computer Vision, vol. 2 (1999), pp. 1033–1038.iSSN: null
6. M. Gumin, mxgmn/WaveFunctionCollapse (2020). original-date: 2016-09-30T11:53:17Z.

https://github.com/mxgmn/WaveFunctionCollapse
7. sol, s-ol/gpWFC (2020). original-date: 2018-05-14T15:25:21Z. https://github.com/s-ol/gpWFC
8. sylefeb, sylefeb/VoxModSynth (2020). original-date: 2017-07-23T04:49:33Z. https://github.com/sylefeb/VoxModSynth
9. S. Leffler, sdleffler/collapse (2020). original-date: 2016-10-25T21:42:16Z. https://github.com/sdleffler/collapse
10. Available: https://twitter.com/mewo2/status/789167437518217216
11. D. Long, Visual Procedural Content Generation with an Artificial Abstract Artist. https://www.academia.edu/34062512/Visual_Procedural_Content_Generation_with_an_Artificial_Abstract_Artist
12. Doodle Insights #19, Logic Data Generation (feat. WFC made easy) (2017). https://trasevol.dog/2017/09/01/di19/
13. crawl/crawl (2020). original-date: 2014-08-17T10:39:37Z. https://github.com/crawl/crawl
14. GDC Vault. https://gdcvault.com/login
15. M. Kleineberg, marian42/wavefunctioncollapse (2020), original-date: 2018-10-25T21:48:08Z. https://github.com/marian42/wavefunctioncollapse