# Procedural Game Content Generation Using the Wave Function Collapse Algorithm*

Martin Dinja

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

xdinja@stuba.sk

30. september 2022

## Contents

**1 Introduction** **1**

**2 Theory** **2**

**3 Implementation** **2**

3.1 Preparation . . . . . 2

3.2 Infrastructure . . . . . 3

3.3 Algorithm . . . . . 3

3.4 GUI . . . . . 4

3.5 Modifications . . . . . 4

**4 Results** **4**

**5 Comparison with other methods** **4**

**6 Conclusion** **5**

**Abstract**

This thesis deals with the problem of generating game content using the Wave Function Collapse algorithm. The algorithm is described in detail and its application to the generation of game content is demonstrated. The thesis also contains a comparison of the algorithm with other methods of generating game content.

## 1 Introduction

Procedural content generation (PCG) is a general term for a system that follows some patterns and generates an output based on those patterns. Its use case is

*Semester project in the subject Methods of engineering work, year 2022/23, management: Name Surname

generating assets or content that would be too time-consuming to create manually. PCG is mainly connotatively tied to game content generation, but its use cases can be more creative. Most PCG systems use game-specific assets with game-specific rules and algorithms to generate their content. A more general application fit for a wide range of games is the Wave Function Collapse algorithm developed by Maxim Gumin [Mat17]. Wave Function Collapse is a greedy PCG algorithm based on the concept of collapsing a wave function, which is a mathematical representation of a quantum state. This method can generate a large, high-quality, consistent output from a small set of input patterns. It can be used in various applications, although its most commonly used for generating 2D tilemaps for games. It can also generate 3D models, music, poetry, and more. WFC's output can only be as good as its input, so it's essential to have a good set of input patterns and rules. My goal in this thesis is to create a simple, configurable application that will use the WFC algorithm to generate tilemaps for games, given a user-defined set of input patterns and rules.

# 2 Theory

As mentioned in the introduction [1], the Wave Function Collapse algorithm is a greedy PCG algorithm based on the concept of collapsing a wave function, which is a mathematical representation of a quantum state. A function starts in a superposition of values and collapses when that function is measured. The result of the measurement is the value of the function at that point. So applying that to the PCG of a 2D tilemap, first, we need to define a set of input patterns and their rules concerning each other. Then the algorithm will assign each tile a superposition of all possible patterns. After prepending the superpositions, the algorithm will assign one random pattern to a tile with the least superpositions, also known as the least entropy, effectively collapsing its wave function, which is also why it is called a collapse algorithm. Then we reevaluate all the affected tiles and their possible patterns and remove the ones that don't fit the rules. Then we repeat the process until there is no more entropy in the tilemap.

# 3 Implementation

## 3.1 Preparation

JavaScript was chosen as the language to implement the WFC algorithm primarily for its workflow and comfort. And for the GUI, I chose p5js, a JavaScript library for creating graphics and animations. It has a simple and easy-to-use API, and it's fast enough for this project.

After deciding on the language and the library, I started by creating tile images and developing a rule system for them. The rule system, rather than detailing every relation to every possible rotation of every tile, uses an array of four numbers representing a connection type for each side of the tile. `[top, right, bottom, left]`. Only the same connection types are allowed to connect to each other.

With that, we can start implementing the infrastructure for the algorithm.

## 3.2 Infrastructure

The infrastructure of the project is the core of the algorithm. It makes it easier for the central part of the algorithm to work with the tilemap. It also contains the main functions of the algorithm.

The infrastructure consists of many functions aiding the algorithm. The most important ones are the ones that work with the tilemap.

The configuration with the rules and the tile images is stored in JSON, detailing each tile's name, picture path, and allowed connections.

Foremost, we need to parse all the patterns and their rules into a format with which the algorithm can interface. We do this by going through each pattern, rotating it, and writing each rotation into a hash map in the form `ct1_ct2_ct3_ct4: [name, img, connections:[], rotate]`. With that, we get a hash map of all the possible rotations of all the patterns.

From the hash map, we create a two-dimensional array of objects which contain the tile's superposition, entropy, and collapse state. The superposition is an integer array of all assignable patterns, the entropy is the number of superpositions, and the collapse state is a boolean value that tells us if the tile is collapsed or not.

Then we only have smaller helper functions for getting a tile's entropy, getting the least entropy tile, collapsing a tile, reevaluating a tile's rules, examining the validity between two tiles, and checking if the tilemap has entirely collapsed.

When updating a cell with the `updateCell` function, we first grab all of the neighboring tiles and their superpositions. Then we go through each neighbor, filtering out the states that fit with none of the neighbors' states. And finally, if the cell remains with only one pattern, we collapse it.

Rule checking is especially vital to the algorithm, even though it's very straightforward. In the `isRuleFollowed` method we take in three parameters, state A, state B, and the direction in which A connects to B. If the connection type of A in the direction is the same as the connection type of B in the opposite direction, then the rule has been followed. In addition to more specific rules, we check if B's tile can be connected to A's tile. To ensure that even if the connection types are the same, tile A still needs to allow connection to tile B, preventing non-compatible tiles from connecting. For instance, if you don't want to prohibit the same tile from connecting to itself.

Other GUI-oriented functions are described in the GUI section [3.4].

Now let's go over the central part of the algorithm.

## 3.3 Algorithm

The central part of the project is the algorithm itself. It uses the helper, interface, and infrastructure functions to work with the tilemap. We start by parsing the JSON file into a hash map and preloading all the needed tile images. Then we create the tilemap and fill each tile's state array with all possible states. Then we start the main loop of the algorithm. In the loop, we get the tile with the least entropy, collapse it, and reevaluate all the affected cells. Evaluating the cells is done by updating the states of each neighbor of the collapsed cell, then reevaluating the states of their neighbors, and so on, until we reach a point where cell states don't change any more. Finally, we check if all of the cells are collapsed, and if they are, the algorithm halts. Otherwise, we repeat the loop.

### 3.4 GUI

The GUI of the project is a simple interface for the user to view the satisfying progression of the algorithm. Its core is a p5js instance that draws the tilemap. Here we only have two functions. The most important one is the `drawGrid` function, which draws the grid. Second is the `drawImg` function, which draws an image given the coordinates, size, and angle.

There are also different options for how and what the GUI will draw. We can disable the constant drawing and instead draw it only when the algorithm has finished. We can also opt-in to draw every possible cell state of a non-collapsed cell.

### 3.5 Modifications

The algorithm is very simple and straightforward, but it can be modified to fit different needs. For a more controlled distribution of tile types we can assign a weight value to each type. The weight value will be used to determine the likelihood of a tile being collapsed into a certain pattern.

We can also expand WFC with generic search, giving it the ability to generate levels targeting specific play experiences. This is the topic of a paper released by Raphael Bailly and Guillaume Levieux [BL22].

There is also a way to augment the algorithm with the Growing Grid neural network for procedural map generation [NMBP20]

We can take control to another level and introduce more constraints to the algorithm making the output seem more Human-Designed. As detailed in the paper by Darui Cheng, Honglei Han, and Guangzheng Fei [CHF20].

## 4 Results

With the algorithm implemented, we can now generate tilemaps. Starting off with a simple example (Figure 1).

Figure 1: Road Tiles

Defining the rules for these tiles, we can generate a 10 by 10 tilemap with the following result (Figure 2):

## 5 Comparison with other methods

When it comes to procedurally generating some content, there are countless of other methods than WFC to choose from. Some of them are more suited for more specific tasks, while others are more general. Some are faster, while others are slower. Some are more complex, while others are simpler. Comparing them is a difficult task, since they have wildly different approaches and goals. But we can still compare WFC to some of the more related methods.

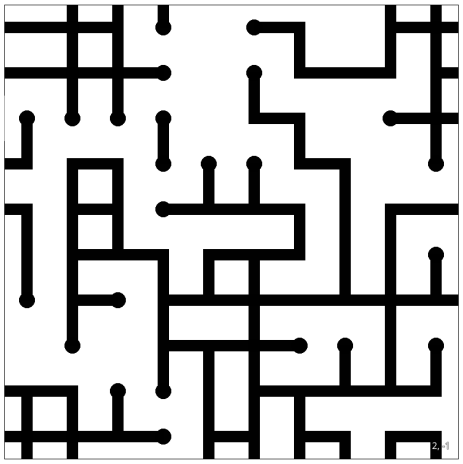

Figure 2: Road Output

As previously mentioned WFC is a constraint-based algorithm. This makes it very configurable, and very deterministic, but it's not very interactive. A very similar algorithm is Model Synthesis, which is a constraint-based algorithm mainly used for generating 3D models. Paul Merrell, the creator of model synthesis has compared the two algorithms in his paper [Mer21].

> *Model synthesis and WFC use nearly the same algorithm and produce similar results. WFC picks cells in a different order and does not modify in blocks. This causes the algorithm to fail more on some large models.*

Another algorithm that is very similar to WFC is the Cellular Automata algorithm. Cellular Automata's model consists of a grid of cells, each of which has a state. The state of a cell is determined by the state of its neighbors. Each new generation is created according to a set of rules. The rules are usually simple, and the algorithm is very fast, but it's not very controllable or deterministic. Its main use case is in generating terrain like caves, rivers, and mountains. It is the foundation for the world generation of Minecraft.

# 6 Conclusion

In this thesis, we gave you a brief introduction to the Wave Function Collapse algorithm and its implementation.

# References

[BL22] Raphael Bailly and Guillaume Levieux. Genetic-wfc: Extending wave function collapse with genetic search. *IEEE Transactions on Games*, pages 1–10, 2022.

[CHF20] Darui Cheng, Honglei Han, and Guangzheng Fei. Automatic generation of game levels based on controllable wave function collapse

Figure 3: Cellular Automata Cave Example

algorithm. In Nuno J. Nunes, Lizhuang Ma, Meili Wang, Nuno Correia, and Zhigeng Pan, editors, *Entertainment Computing – ICEC 2020*, pages 37–50, Cham, 2020. Springer International Publishing.

[Mat17] Maxim Matsievski. Wave function collapse. `https://github.com/mxgmn/WaveFunctionCollapse`, 2017.

[Mer21] Paul C. Merrell. Comparing model synthesis and wave function collapse. 2021.

[NMBP20] Tobias Nordvig Møller, Jonas Billeskov, and George Palamas. Expanding wave function collapse with growing grids for procedural map generation. In *International Conference on the Foundations of Digital Games*, FDG '20, New York, NY, USA, 2020. Association for Computing Machinery.