# Combining Constructive Procedural Dungeon Generation Methods with WaveFunctionCollapse in Top-Down 2D Games

Pedro Probst Minini
*Department of Applied Computing*
*Universidade Federal de Santa Maria (UFSM)*
Santa Maria, Brazil
ppminini@inf.ufsm.br

Joaquim V. C. Assunção
*Department of Applied Computing*
*Universidade Federal de Santa Maria (UFSM)*
Santa Maria, Brazil
joaquim@inf.ufsm.br

***Abstract*—Recent development of new procedural generation algorithms may provide an expanded generation pipeline to game programmers and designers. This work provides insight into how to combine the innovative WaveFunctionCollapse (WFC) with traditional and well-known constructive dungeon generation algorithms to create architecturally interesting and diverse environments, while discussing the strengths and weaknesses of utilizing WFC as part of the dungeon generation process and how the algorithms may interact with each other. Finally, we briefly show how our generation system works and outline possible next steps for our work.**

***Index Terms*—procedural content generation, procedural dungeon generation, wave function collapse**

## I. Introduction

It is well known that *procedural content generation* (PCG) provides a greater degree of replay value in games – besides the consequence of cutting production costs. Depending on the game genre, PCG may be utilized densely or scarcely, and in a variety of ways: in the generation of items, quests, environments, characters, and further on. It has become common for PCG to be present from AAA titles to independent (indie) games.

However, PCG is more densely observed in the indie gaming community, mainly because it is composed of smaller development teams. In recent years, *roguelikes* and similar genres have been responsible for popularizing and thus pushing the boundaries of procedural generation even further. For instance, *Dwarf Fortress* – a colony-builder/sandbox game – simulates the whole history of a fictional world as a way to create unique storytelling [1]. The roguelike *Caves of Qud* makes creative use of procedural generation of environments and narrative [2] to simulate a *dying earth* themed world, with its own ruins, ancient sultans, and cults.

Due to its broad research spectrum, this work focuses on a single aspect of PCG: *procedural dungeon generation* (PDG). Specifically, we show – in a simple way – how to expand a generic implementation of the *WaveFunctionCollapse*[1] (WFC) algorithm with traditional and well-known constructive generation methods. While dungeon generation pipelines are constructed and polished through the whole development of a game, this work may shed some knowledge on how to creatively combine different generation algorithms to create interesting top-down 2D environments.

[1]Originally developed by Maxim Gumin and publicly available at https://github.com/mxgmn/WaveFunctionCollapse

## II. Related Work

In *Roguelike Celebration* 2019, Brian Bucklew [3] described his dungeon generation pipeline in *Caves of Qud* – which was the first commercial game to adopt WFC. Unfortunately, only a few works in academia are related to using WFC.

In one of the first published papers on WFC, Karth and Smith [4] explained the algorithm and its strengths, while formulating it as an answer set programming (ASP) problem. Furthermore, they showed WFC being used to generate levels in games and even poetry.

Scurti and Verbrugge [5] show how can WFC be modified to create interesting randomized paths for non-playable characters (NPCs) in games.

Kim *et al.* [6] proposes a graph-based WFC algorithms instead of its common grid-based implementation, to facilitate the use of PCG in graph space and 3D worlds efficiently.

Sandhu, Chen and McCoy [7] describe an implementation of WFC integrating specific design constraints to generate levels, through non-local constraints and dynamic weighting, thus providing more control of the developer over the algorithm.

Finally, our work is unique in the sense that we show how to use a standard implementation of WFC (adapted to game tiles) in a practical way conjunctively with other algorithms commonly used in top-down 2D games – expanding on Bucklew's presentation [3].

## III. Methods

There are countless PDG techniques publicly available online. Valuable information can be acquired through game

websites, wikis[2], social networks[3] and development blogs[4]. Although "dungeon" is a term utilized extensively, they may not always represent a secluded, closed space: informally, PDG methods can also comprise the generation of less claustrophobic environments like forests, valleys, and canyons.

In this work, the selected algorithms are summarized in Table I. Following Togelius' *et al.* PCG taxonomy [8], all these algorithms are stochastic and generated during the runtime of the game. They also have some degree of control through parametrization, although this can be implementation-dependent.

TABLE I
BRIEF COMPARISON BETWEEN CHOSEN ALGORITHMS

| Algorithm | Gen. Method | Input | Output | Conectivity | Complexity |
|---|---|---|---|---|---|
| Drunkard Walk | Constructive | Filled | *TDCL* | Depends | Low |
| Cellular Automata | Constructive | Chaotic | *TDCL* | No | Low |
| BSP Dungeon | Constructive | Filled | *TDML* | Yes | Medium |
| Digger | Constructive | Filled | *TDML* | Yes | Medium |
| WFC | Search-based | Any | Input-depend. | No | High |

A *filled* input corresponds to a list of tiles representing a map (or dungeon) where all the tiles are of a single, solid type; a *chaotic* input represents a mix of tile types. The output of each algorithm can be top-down cavern-like (*TDCL*), top-down mansion-like (*TDML*) – separated rooms typically connected with corridors – (see [9]) or locally similar to the input (*input-dependent*). "Complexity" is related to the implementation difficulty.

Our repository with the implementation of each algorithm as part of a game prototype is publicly available on GitHub[5].

### A. Drunkard Walk and Cellular Automata

The *Drunkard Walk* and *Cellular Automata* (CA) [10] algorithms can be used together to create cave systems and forests. As the Drunkard Walk algorithm generates chaotic maps, and the Cellular Automata algorithm is specifically made to work with chaotic inputs, they are naturally associative.

The Drunkard Walk is essentially a random walk algorithm. It uses agents that "dig" the filled map, generating a single cave (when the agents are started in the same position) or a cave system (when agents are started in different positions). The life span of each agent, and the direction followed at each step, are randomly determined from a chosen range, and the number of agents is increased with each iteration of the algorithm, which depends on the number of floor tiles desired. In the algorithm mode for the generation of a cave system instead of a single cave, connectivity between regions is not ensured.

The Cellular Automata algorithm for generating dungeons works based on a chaotic map (typically between $40\%$ and $50\%$ of floor tiles) as an input. Based on user-defined rules, the map is "smoothed" in $n$ iterations; the greater the number of iterations, the greater the smoothing suffered on the map.

The neighboring tiles to be considered can follow *Moore's neighborhood* (orthogonal and diagonal neighbors) or *von Neumann's neighborhood* (only orthogonal neighbors). The programmer has flexibility in making his own rules. However, the more rules are inserted, the slower the algorithm will run; depending on the size of the generated map, it can harshly impact performance.

### B. BSP Dungeon

The *Binary Space Partitioning (BSP) Dungeon* algorithm is one of the most commonly used TDML algorithms and it is based on the subdivision of space and the subsequent addition of rooms in the generated spaces, based on a BSP tree.

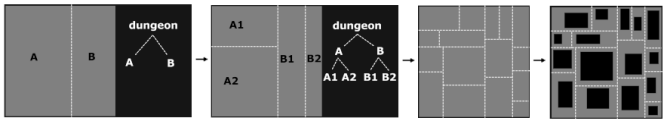



Fig. 1. Visual representation of the BSP dungeon generation algorithm. Image adapted from *RogueBasin.*

After the insertion of the rooms – which can be of any form –, the room list can be reordered by position, size, among other attributes. The connections between the rooms are usually made sequentially; therefore, the order of the rooms in the list will govern how the rooms will be connected, changing the final result.

### C. Digger/Tunneler

In addition to the BSP Dungeons algorithm, *Digger* [11] (or *Tunneler*) is another algorithm used to generate TDML maps. In this algorithm, the generated rooms are typically derived from a centrally located room. At each iteration, a random edge index of the room being analyzed is chosen; and from this index, another room is generated in the same direction as the edge, chosen at a randomly determined distance. After that, the two rooms are connected by a "narrow room", defining a corridor. The algorithm ends when the maximum number of iterations has been reached, which potentially indicates the impossibility of inserting rooms of the requested size.

Despite sharing a common name, the Digger algorithm tends to be severely modified between implementations, albeit the core idea remains the same.

### D. WaveFunctionCollapse

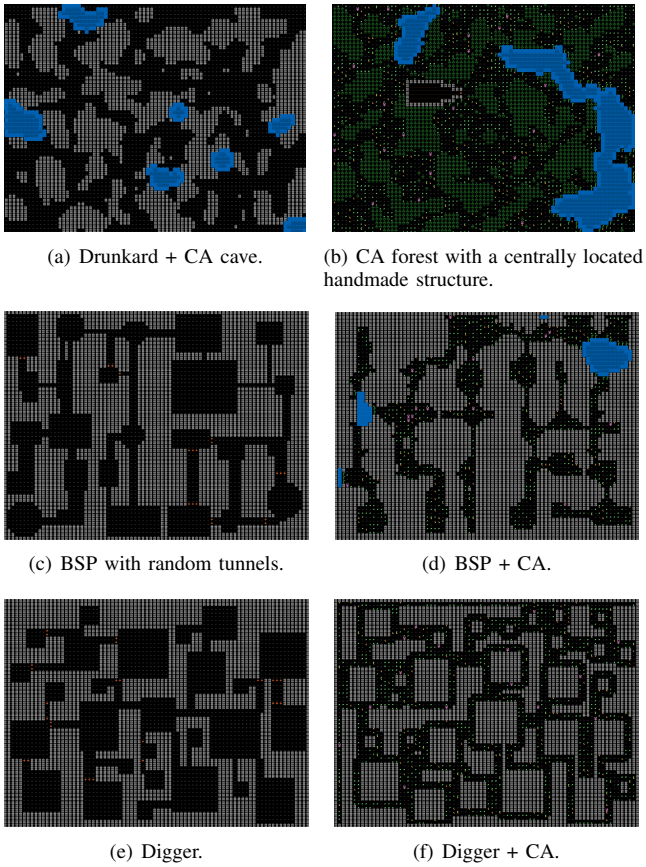
The *WaveFunctionCollapse* algorithm (Fig. 3) was the only search-based algorithm chosen for comparison, due to its innovative aspects and increasing use in the industry. WFC takes a small input (image or tile map) and generates a larger output which is locally similar to the input. Originally made to work with pixel values on images, it was adapted in this work to accept generic tile types. Our implementation derived from [12].

[2]See http://pcg.wikidot.com/ and http://www.roguebasin.com/

[3]There is a *subreddit* where various developers post their experiments with content generation. See https://www.reddit.com/r/proceduralgeneration/

[4]Since 2013, *Cogmind* developer Josh Ge maintains a blog where he goes in detail about his experience with different technical aspects of game development. See https://www.gridsagegames.com/blog/

[5]See https://github.com/pprobst/tcc-ufsm-2020



(a) Drunkard + CA cave.

(b) CA forest with a centrally located handmade structure.

(c) BSP with random tunnels.

(d) BSP + CA.

(e) Digger.

(f) Digger + CA.

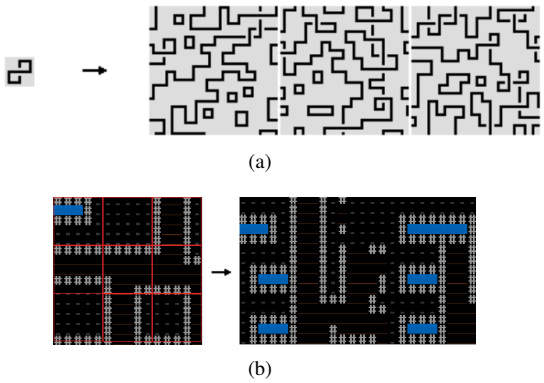
Fig. 2. Six 80x60 outputs of the constructive algorithms. Note how details like vegetation, structures and doors (orange '+' characters) can be inserted as a post-process step. CA can also be combined with essentially *all* algorithms to produce distinct results, like the BSP-derived ruins in (d) and the "inverted digger" in (f).

Essentially, WFC is a complex solver [4] – each cell of fixed size on the map initially has the possibility to accept all the tile patterns generated from the partitioned input (which are reflected, rotated, and so forth), and from the *adjacency rules* (based on similarity in the Overlapping model [12]) the set of patterns is reduced until the cell is "collapsed", i.e., it has only one possible pattern. The changes are propagated from the current cell being analyzed to the neighboring cells, which also have their sets of possibilities reduced; as an analogy, this step works similarly to a game of *Sudoku*.

The algorithm ends when all the cells on the map are collapsed; otherwise, the algorithm arrives at a contradiction – in this case, it can start over or backtrack, depending on the implementation. Each collapsed cell is then translated as output. WFC is highly dependent on the input structure. Thus, the quality of the generated map mainly depends on the input.

It is important to note that WFC is not a *panacea*. While experimenting with WFC, Bucklew [3] encountered some drawbacks:

1) Homogeny: there is no inherent large structure (e.g. a village of rectangular houses may not have a cathedral).
   - Solution: as a pre-processing step, select a large region on the map and run WFC inside it to generate internal architecture.
2) Overfitting: adding more detail often results in overfitting small details, reducing the variability of the output.
   - Solution: inserting the small details (doors, furniture, etc.) as post-processing steps instead of adding them on the input.
3) No connectivity: WFC has no way to ensure connectivity between regions.
   - Solution: run a connection algorithm as a post-processing procedure.



(a)

(b)

Fig. 3. (a) shows an output of the original WFC by Maxim Gumin applied to bitmap images, taking into account the pixel values. (b) shows an output of our implementation of WFC applied to tiles, taking into account the tile type (wall, floor, water, etc.) and using a tile size partition of 5 (the input totals 15x15 tiles in size). The output is cropped from a larger 80x60 map.

Comparing to our work, all the problems above were more prominent when smaller tile sizes were chosen to partition the input. Also, if there are identifiable larger structures in the input, a larger tile size may be needed to partition the input, otherwise the larger structures will simply be incorrectly represented in the output (e.g. discontinuities). The problem with using larger tile sizes is the decrease of output variability, but it is a way to circumvent the limitations of WFC regarding large structures. Interestingly, with larger tile sizes, the "mix-and-match" essence of WFC becomes clear.

Because of such problems, WFC (in the context of top-down 2D games) may be better used as an algorithm to generate interesting internal architecture instead of large, complex structures. This reinforces the notion that WFC is best used when combined with other PDG algorithms, i.e., WFC may be utilized as just another step in the generation process.

## IV. Generation System

To test the dungeon generation methods, we needed a simple yet effective system. In our system, a *Map* structure represents all the visual information contained in a dungeon, storing tile and entity (player, mobs and items) information. A Map can be created using the *Map Generator* structure, which in turn can call the various generation pipelines implemented.

*Base Pipelines* are used to generate "basic" environments utilizing the various algorithms discussed in Section III. *Specific Pipelines* are functions that use the Base Pipelines to generate more complex dungeons (Fig. 4).

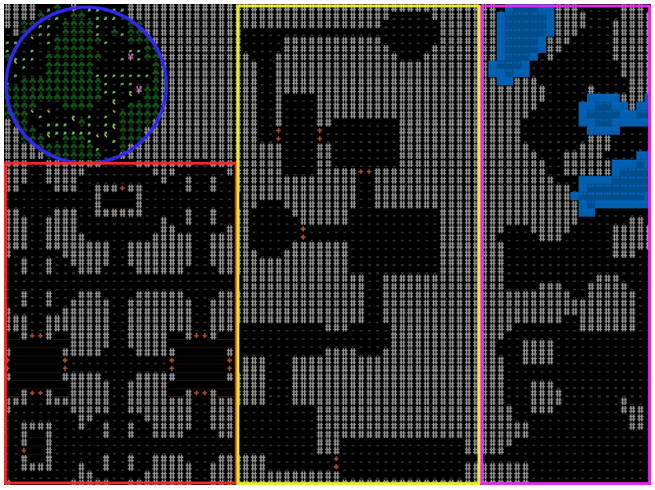


Fig. 4. Example output (80x60) created by a specific pipeline. Blue region: a small forest generated with Cellular Automata (CA). Red region: this is a case where a larger tile size partition was used to generate exterior architecture with WFC. Yellow region: mostly vertically connected rooms created with BSP Dungeon. Pink region: cave created with Drunkard Walk and subsequent steps of CA (note the distinction between shallow water and deep water, created purely through rule manipulation).

Note that various algorithms utilized (Table I) do not assure total connectivity, and by combining different generation methods together we run a greater risk of creating inaccessible regions. The *connectivity problem* can always be solved as a post-process pass; in our case, we simply utilize the *flood fill* algorithm to detect isolated regions on the map. After collecting the different regions in a list (which may be sorted according to one's needs), we detect the *edges* of each region and calculate the closest point between two regions, and then a connecting tunnel is carved on the map. The result can be easily seen on Fig. 4, as the tunnel between the yellow and pink regions (bottom right).

Finally, in Fig. 5 we show three outputs of WFC being used to generate internal architecture for rooms already created.

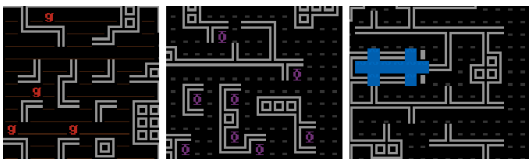


Fig. 5. Cropped WFC outputs used as internal architecture for rooms generated with the Digger algorithm. Because WFC can technically accept any tile, it is also possible to insert mobs (red characters) and general furniture (magenta characters), thus creating a basic spawning system – which may be interesting for small treasure rooms (vaults).

## V. Conclusion

Through experimentation and simple generation pipelines, our work showed how can different PDG methods be combined to create interesting top-down 2D environments through localized generation with WFC. While most roguelikes depend on a single Base Pipeline for each dungeon level and prefabricated (handmade) structures, we encourage developers to try and experiment with mixed pipelines to generate more diverse and/or organized maps.

Our next steps may include – besides the further development of our pipelines – the exploration of other aspects of PCG, such as procedural generation of narrative, culture, societies and politics. Mark R. Johnson's roguelike *Ultima Ratio Regum* [13] is a prime example of exploration of such topics, in which they all act intertwined through Qualitative Procedural Generation (QPG). Furthermore, the algorithms here presented can also be translated to 3D, including WFC [14]. The use of Artificial Intelligence (AI) in PCG is also in crescent use, as recent experiments with PCG via reinforcement learning [15] and through generative playing networks [16] show that it is a promising study area, and may also be part of our future work.

## References


[1] B. in het Veld, B. A. Kybartas, and R. Bidarra, "Procedural generation of populations for storytelling," 2015.

[2] J. Grinblat and C. B. Bucklew, "Subverting historical cause & effect: generation of mythic biographies in caves of qud," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pp. 1–7, 2017.

[3] C. B. Bucklew, "Dungeon generation via wave function collapse." https://youtu.be/fnFj3dOKcIQ, 2019. Accessed: 2019-12-09.

[4] I. Karth and A. M. Smith, "Wavefunctioncollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pp. 1–10, 2017.

[5] H. Scurti and C. Verbrugge, "Generating paths with wfc," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

[6] H. Kim, S. Lee, H. Lee, T. Hahn, and S. Kang, "Automatic generation of game content using a graph-based wave function collapse algorithm," in *2019 IEEE Conference on Games (CoG)*, pp. 1–4, IEEE, 2019.

[7] A. Sandhu, Z. Chen, and J. McCoy, "Enhancing wave function collapse with design-level constraints," in *Proceedings of the 14th International Conference on the Foundations of Digital Games - FDG '19*, ACM Press, 2019.

[8] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

[9] B. M. F. Viana and S. R. dos Santos, "A survey of procedural dungeon generation," in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, IEEE, oct 2019.

[10] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," 2010.

[11] N. Shaker, A. Liapis, J. Togelius, R. Lopes, and R. Bidarra, "Constructive generation methods for dungeons and levels," in *Procedural Content Generation in Games*, pp. 31–55, Springer, 2016.

[12] S. Sherratt, "Procedural generation with wave function collapse." https://gridbugs.org/wave-function-collapse/, 2019. Accessed: 2020-02-10.

[13] M. R. Johnson, "Towards qualitative procedural generation." https://youtu.be/Mk-TmpSUb54, 2016. Accessed: 2020-06-24.

[14] O. Stålberg, "Wave function collapse in bad north." https://youtu.be/0bcZb-SsnrA, 2018. Accessed: 2020-06-24.

[15] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," *arXiv preprint arXiv:2001.09212*, 2020.

[16] P. Bontrager and J. Togelius, "Fully differentiable procedural content generation through generative playing networks," *arXiv preprint arXiv:2002.05259*, 2020.