Extend Wave Function Collapse Algorithm to Large-Scale Content Generation

1st Yuhe NIE
Computer Science and Engineering
Southern University of Science and Technology
Guangdong, China
nieyh2019@mail.sustech.edu.cn

2nd Shaoming ZHENG
Computer Science and Engineering
Southern University of Science and Technology
Guangdong, China
zhengsm2023@mail.sustech.edu.cn

3rd Zhan ZHUANG
Computer Science and Engineering
Southern University of Science and Technology
Guangdong, China
11811721@mail.sustech.edu.cn

4th Xuan SONG*
Computer Science and Engineering
Southern University of Science and Technology
Guangdong, China
songx@sustech.edu.cn

Abstract—Wave Function Collapse (WFC) is a widely used tile-based algorithm in procedural content generation, including textures, objects, and scenes. However, the current WFC algorithm and related research lack the ability to generate commercialized large-scale or infinite content due to constraint conflict and time complexity costs. This paper proposes a Nested WFC (N-WFC) algorithm framework to reduce time complexity. To avoid conflict and backtracking problems, we offer a complete and sub-complete tileset preparation strategy, which requires only a small number of tiles to generate aperiodic and deterministic infinite content. We also introduce the weight-brush system that combines N-WFC and sub-complete tileset, proving its suitability for game design. Our contribution addresses WFC's challenge in massive content generation and provides a theoretical basis for implementing concrete games.

Index Terms—Wave Function Collapse, Procedural Content Generation, Tessellation, Constraint Satisfaction, Algorithm Optimization, Game Intelligence, Level Design

I. INTRODUCTION

VOLUME of the game is growing, and developers are constantly looking for new ways to create immense virtual worlds. Procedural Content Generation (PCG) is a technique that can generate vast and varied game worlds without requiring developers to create every aspect of the environment manually. Wave Function Collapse (WFC), proposed by Maxim Gumin [1], is a popular PCG algorithm that can generate textures, 2D and 3D entities, and other abstract forms and is widely used in game [2], [3], [4].

However, the current WFC algorithm suffers from exponential time complexity, which limits its practical application in large-scale or real-time infinite scene generation. As the size of the generated content increases, the time and number of conflicts WFC encounters will also increase, making the

* Corresponding author

performance overhead worse. To address this challenge, this paper proposes a Nested Wave Function Collapse (N-WFC) algorithm framework that uses several internal WFCs (I-WFC) with fixed generation sizes nested within a exterior generation process. By maintaining constraints between adjacent content that I-WFC generates, N-WFC can generate large-scale or infinite scenes with polynomial time complexity.

To ensure that N-WFC generates accepted solutions, reduces conflicts, and avoids backtracking after I-WFC generation, we introduce the mathematical definitions of complete and sub-complete tilesets. The complete tileset requires a large number of tiles to satisfy the constraints exhaustively, while the sub-complete tileset can achieve similar effects using fewer tiles and is more practical for game development. We prove mathematically that both complete and sub-complete tilesets can generate infinite and aperiodic tessellation. Importantly, every result generated by I-WFC under the N-WFC framework is deterministic and will not require backtracking.

We conduct a series of experiments using sub-completions with varying numbers of definitions to test the efficacy of N-WFC. The results show that N-WFC significantly reduces the time required for content generation compared to WFC. Additionally, N-WFC is highly customizable and extensible, allowing for creating tilesets with different design properties using a weighted brush system. We demonstrate how the Carcassonne game can be used as an example to implement N-WFC and make design decisions.

In summary, this work solves the time complexity and conflict problems of WFC in content generation, enabling deterministic infinite content generation for both 2D and 3D game scene generation. N-WFC can be combined with other heuristics [5] or learning algorithms [6] to assist in the realization of design strategies, extending the performance

and uses-scope of WFC and opening up new possibilities for procedural content generation.

II. RELATED WORK

A. PCG Algorithms for Scene Generation

Procedural content generation (PCG) in scene generation has been a popular topic since the emergence of roguelike games [7]. Rogue [8], developed in 1985, was the first game that used PCG. Since then, numerous PCG techniques for scene generation have been proposed.

The goal of PCG on scenes is to add randomness to scene generation, making each experience different. However, the generated scene should still be logical and cater to the designer's ideas. As game developer Herbert Wolverson [9] explained:

> The randomness does not define the above games. It is fed into an algorithm that generates something that approximates what you want to get but ensures that it is different every time.

Early scene generation techniques focused on 2D dungeon-shaped games. Algorithms like Random Room Placement or Binary Space Partition (BSP) [10] focused on the creation of logical and connected rooms. Later, Cellular Automata (CA) [11], and Drunkard Walk [12] was introduced to give the generated scene irregularity, making it look more like nature's irregularities. Voronoi Diagrams [13] and Perlin Noise [14] pushed PCG towards generating larger city or continental plates. These algorithms use randomness and noise and add post-processing algorithms to ensure reasonable scenes. These techniques are suitable for 2D and 3D natural terrain generation.

However, these early algorithms were not suitable for strong-design program scene generation. Maxim Gumin offered two algorithm strategies, Wave Function Collapse (WFC) [1] and MarkovJunior [15], which make the generation result in more of a product of design intuition than nature's randomness. In WFC, generation is described as a process of satisfying a binary constraint of connected nodes. In MarkovJunior, the technique is based on pattern matching and constraint propagation. Randomness no longer serves to generate itself but instead makes probabilistic choices based on design.

B. Wave Function Collapse

Wave Function Collapse (WFC) has gained significant attention since its introduction, as it generates output containing only patterns (tiles) similar to the input, making it suitable for generating objects for design purposes. Many game applications use WFC for limited scene generation, such as Caves of Qud [2] with WFC-created ruins, Bad North [3] for generating scenes and maintaining the navigation mesh for arranging the Viking war, and Townscraper [4] for building a town underlying WFC. Additionally, reinforcement learning teams used WFC to generate arenas for their agents [16].

Several studies have optimized the WFC algorithm, including expressing it as a Constraint Satisfy Problem (CSP) and applying the AC-3 algorithm to check consistency, as described by Paul C. Marrell [17]. Karth *et. al.* [18] examined backtracking and combined different heuristic algorithms in WFC, while Bailly *et. al.* [5] mixed the genetic algorithm with WFC to guide generated content's novelty, complexity, and safety. Kim *et. al.* [19] proposed a graph-based WFC that supports navigation in 3D scene generation. However, the problem of the exponential time complexity of the WFC algorithm still exists, limiting its ability to generate a large range of content. This problem also causes backtracking, making it unsuitable for commercial games. Marian Kleineberg [20] tried to use WFC to create an experimental game, the Infinite City, but noted that conflicts and backtracking could lead to regenerating existing parts, further limiting its application in commercial games.

C. PCG Serves as Level Design

PCG is used in various aspects of game design, including level design, which primarily serves two purposes. First, it reduces the burden of creating different levels while maintaining the game's desired quality. Second, it creates varied experiences for players, encouraging them to use internal strategies to face unforeseen circumstances rather than relying on memorization.

PCG on level design abstracts the game mechanics, such as difficulty rhythms, mood experiences, and task flows. There are several approaches to PCG on level design. *Constructionist and Grammatist* algorithms use pre-defined content chunks, such as grammar rules, or randomly place chunks one after another. *Constraint Driver* [1], [15] algorithms design specific constraints and use a constraint solver to find potential solutions. *Optimizer and Learner* [21], [22] algorithms respond to designing fitness functions and constraints of level features. They learn latent information and tune parameters independently to meet their desired outcomes.

For algorithms that assist in level design, whether explicit or implicit, they should expose controllable parameters and specific decision ideas for designers to use [23]. These parameters can be used for various needs, such as curated design, player preferences, and adaptive difficulty.

III. WFC AND N-WFC ALGORITHM

A. WFC Introduction

The Wave Function Collapse Algorithm (WFC) has two implementation strategies, the *Simple Tiled Model* and the *Overlapping Model*. These two models only differ in forming tilesets from input samples. Both of them contribute equally to the algorithm's core, with the Overlapping Model generally regarded as more convenient due to its ability to extract tiles and their connectivity from the input sample image automatically.

1) Simple Tiled Model: The Simple Tiled Model can be either created manually or generated by the program. For a 2D game, as Fig. 1 shows, it works by handcrafting the tileset and setting up the connectivity of the four edges of the tiles using binary constraints.
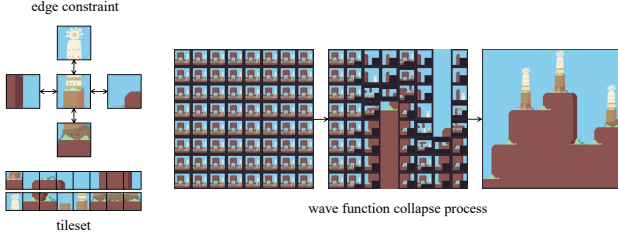
Fig. 1. Simple Tiled Model Demonstration (An interactive demo provided by Oskar Stålberg [24]. The tileset contains the edge adjacencies information of each tile. WFC starts by creating a grid with unobservable (uncertain) cells. At each iteration, it picks one cell to collapse with one available tile for that cell that minimizes the Shannon entropy and propagates the edge constraints to affected cells. The algorithm terminates when all the cells have collapsed or a conflict is encountered)

2) Overlapping Model: The Overlapping Model can automatically generate a tileset [1] from an input sample image. It extracts the tiles by using convolution to encode the pattern of the local neighborhood of each pixel on the sample image and then classifying them into tiles. The tiles adjacencies are also searched and constructed automatically.

B. Problem Formulation

This paper discusses WFC operates on the simple tiled model since the overlapping model doesn't meet the tileset requirements discussed later. For simplicity, the problem is formulated in the 2D case.

1) Tileset and Edgeset: Let $\mathcal{T}$ be a finite tileset. Each tile $t \in \mathcal{T}$ can be represented as a combination of 4 edges, as $t = (e_n, e_s, e_w, e_e)$ because edges are the only thing that matters in adjacency constraints. $e_n, e_s \in \mathcal{E}_{\mathcal{NS}}$, $e_w, e_e \in \mathcal{E}_{\mathcal{WE}}$. $\mathcal{E}_{\mathcal{NS}}$ and $\mathcal{E}_{\mathcal{WE}}$ are the sets of available edges on the north-south and west-east directions respectively. Depending on different implementations, $\mathcal{E}_{\mathcal{WE}}$ and $\mathcal{E}_{\mathcal{NS}}$ may or may not share some of the available edges. Each edge can be denoted as $e_{dir}(t)$, $dir \in \{n, s, w, e\}$ for a tile t or $e_{dir}(m, n)$, $m \in [1, M], n \in [1, N]$ for a cell with index (m, n).

2) Initialization: WFC executes on an $M \times N$ grid matrix $\mathbf{G}$ where $g_{m,n} \subseteq \mathcal{T}$. A *tiling* of $\mathbf{G}$ by $\mathcal{T}$ is a function $f : \mathbf{G} \leftarrow \mathcal{T}$. Initially, all the cells are in unobservable states. $\forall m \in [1, M], \forall n \in [1, N], g_{m,n} = \mathcal{T}$.

3) WFC Solver: The WFC solver starts looping by executing the following process:

1) Choose a cell $g_{m,n}$ that has the minimum available tiles (Minimum Remaining Value).
2) Randomly pick an available tile for the cell $t \in g_{m,n}$ and collapse this cell into a definite state: $g_{m,n} \leftarrow \{t\}$
3) Propagate the new constraints into adjacency cells of $g_{i,j}$ and reduce tiles that do not satisfy the constraints until all the edge consistencies are maintained.

---

[1]Note that in Gumin's interpretation, the *tiles* in an Overlapping Model are subdivided from the sample image, which is different from the Simple Tiled Model. While the unit of the "tileset" for the constraint solver is actually pattern region or group of tiles. Here we equate *tiles* to pixels for simplicity.

4) Accepted Result: An *accepted* WFC result is defined as[2]:

$$\forall_{m=1}^{M} \forall_{n=1}^{N} |g_{m,n}| = 1$$
$$e_n(m, n) = e_s^*(m - 1, n)$$
$$e_s(m, n) = e_n^*(m + 1, n)$$
$$e_w(m, n) = e_e^*(m, n - 1)$$
$$e_e(m, n) = e_w^*(m, n + 1)$$

5) Time Complexity of WFC: WFC is a constraint satisfaction problem that solves binary edge constraints, which may lead to conflicts during the generation process. It uses backtracking to ensure an accepted solution if it exists and uses AC-3 to optimize and maintain arc consistency. Suppose the number of the tileset is $|\mathcal{T}| = d$, the time complexity of WFC is $O(d^{M \times N} + (M \times N)^2 d^3)$, which grows exponentially with the size of the generation grid.

C. Nested Wave Function Collapse (N-WFC)

The time-consuming problem of WFC can make it challenging to generate large-scale content. To address this issue, we propose Nested Wave Function Collapse (N-WFC), which splits a large-scale grid matrix into smaller *sub-grids* that can be solved more efficiently by using interior WFC and maintaining the constraints of each sub-grid. N-WFC is based on the domain splitting problem of CSP, which divides a large problem into smaller tasks while maintaining the overall constraints. Algorithm 1 shows the pseudocode of N-WFC, and the following section explains how it works.

1) Interior WFC (I-WFC): I-WFC generates a small, fixed-size sub-grid $\mathbf{G}^{sub} \in \mathbb{Z}^2$ with $C \times C$ grid matrix, where C is a constant and typically a small value. I-WFC works similarly to regular WFC, with the only difference being that it may have some pre-constraints propagated from other sub-grids. Specifically, cells in the first row and first column of $\mathbf{G}^{sub}$ may have already been collapsed before starting the I-WFC: $\exists g^{sub}, \forall m \in [1, C], |g_{m,1}^{sub}| = 1, \exists \mathbf{G}^{sub}, \forall n \in [1, C], |g_{1,n}^{sub}| = 1$.

2) Exterior Diagonal Generation Process: To generate an N-WFC result, we split the large-scale grid matrix into sub-grids problem and operated I-WFC to solve the problem. Suppose the entire task contains $A \times B$ sub-grids, the whole tasks contains $M \times N$ cells, where $M = A \cdot (C - 1) + 1$, and $N = B \cdot (C - 1) + 1$. In the later section, we use $\mathbf{G}^{a,b}$ to denote for each sub-grid with index (a, b). For simplicity, the generation process starts from the top-left corner and progresses in a diagonal sequence, layer by layer. During generation, each sub-grid contains the value of its left neighbor's rightmost column and its upper neighbor's last row (if it exists), ensuring that the adjacent edges of adjacent sub-grids are consistent and can be treated as the same edge for overlapping. When all the sub-grids have been generated, each adjacent sub-grid overlaps its first row and first column, forming the N-WFC result.

---

[2]e_{dir}^* denotes the equation will be checked if such cell exists.

3) Accepted Result: We denote each sub-grid as $\mathbf{G}^{a,b}, a \in [1, A], b \in [1, B]$. For the N-WFC framework, an *accepted* N-WFC result will be[3]:

$$\forall_{a=1}^{A} \forall_{b=1}^{B} \mathbf{G}^{a,b} \text{ is an accepted WFC}$$
$$\wedge \forall_{a=1}^{A} \forall_{b=1}^{B} \forall_{m=1}^{C} g_{m,1}^{a,b} = (g_{m,C}^{a,b-1})^* \wedge |g_{m,1}^{a,b}| = 1$$
$$\wedge \forall_{a=1}^{A} \forall_{b=1}^{B} \forall_{n=1}^{C} g_{1,n}^{a,b} = (g_{C,n}^{a-1,b})^* \wedge |g_{m,1}^{a,b}| = 1$$

Algorithm 1 NESTED WFC

Output: returns an *accepted* solution
  create a N-WFC grid matrix $\mathbf{G}$
  $|\mathbf{G}| = (A \cdot (C-1) + 1) \times (B \cdot (C-1) + 1)$
  split the $\mathbf{G}$ into $A \times B$ sub-grid, each $|\mathbf{G}^{sub}| = C \times C$
  for layer in diagonal layer **do**
    for $\mathbf{G}^{a,b}$ in layer **do**
      initial empty sub-grid $\mathbb{G}, |\mathbb{G}| = C \times C$
      $\mathbb{G}_{:,1} \leftarrow (\mathbf{G}_{:,C}^{a,b-1})^*$
      $\mathbb{G}_{1,:} \leftarrow (\mathbf{G}_{C,:}^{a-1,b})^*$
      $\mathbb{G} \leftarrow \text{WFC}(\mathbb{G})$
      $\mathbf{G}^{a,b} = \mathbb{G}$
      paste $\mathbf{G}^{a,b}$ into $\mathbf{G}$, overlapping one edge with $\mathbf{G}^{a-1,b}$
      and $\mathbf{G}^{a,b-1}$ if exist
    end for
  end for
  return $\mathbf{G}$

---

4) Time Complexity of N-WFC: N-WFC split the task into $A \times B$ small tasks, with each sub-grid requiring I-WFC to use backtracking and AC-3, having the time complexity of $O(d^{C^2} + C^4 d^3)$. The upper bound of N-WFC contains no more than $\frac{M \times N}{C^2}$ tasks. Therefore, the overall time complexity of N-WFC is $O(\frac{M \times N}{C^2} d^{C^2} + (M \times N) C^2 d^3)$. Since C^2 is a relatively small constant, N-WFC has only polynomial time complexity. As the range of generated content increases, N-WFC's time cost will become significantly better than that of WFC.

5) Conflict Problem: While N-WFC has better time complexity than WFC under ideal circumstances, it still faces the conflict problem caused by the edge constraints in practice. I-WFC may fail to find a suitable tile that satisfies certain deterministic edge types, resulting in an inability to generate an accepted solution and requiring the diagonal generation process to use backtracking. N-WFC is also less effective on the overlapping model, which constructs edge sets with a large amount of edge types, but with few tiles that satisfy the constraints. Therefore the overlapping model is not discussed in this paper. In Section IV, we introduce two tileset construction methods to solve the conflict problem.

IV. COMPLETE AND SUB-COMPLETE TILESETS

This section introduces two methods for creating tilesets that can be used for aperiodic tessellation and deterministic infinite content generation. These methods require only a limited

---

[3] $(\mathbf{G}_{m,n}^{a,b})^*$ denotes the equation will be checked if such sub-grid exists.
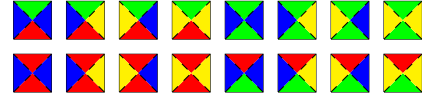

Fig. 2. An Example of the Complete Tileset. Color of the edge represents different edge types. $\mathcal{E}_{\mathcal{NS}} = [\text{red, green}]$ and $\mathcal{E}_{\mathcal{WE}} = [\text{blue, yellow}]$, It traverses all combinations of edge types and satisfies the minimum requirement of the complete tileset. $|\mathcal{T}| = |\mathcal{E}_{\mathcal{NS}}|^2 \cdot |\mathcal{E}_{\mathcal{WE}}|^2 = 2^2 \cdot 2^2 = 16$.

Fig. 3. An example of the Sub-complete Tileset. Color of the edge represents different edge types.

number of tiles and can be designed in 2D or 3D. The *complete* tileset can accomplish tessellation without any conflicts during the WFC process, while the *sub-complete* tileset can overcome conflicts using the N-WFC algorithm.

A. Tileset Definition

To maintain the aperiodic characteristic, we require $\max\{|\mathcal{E}_{\mathcal{NS}}|, |\mathcal{E}_{\mathcal{WE}}|\} \geq 2$, meaning that at least one of the edge sets contains two or more types of edges.

1) Complete Tileset: The complete tileset (Example as Fig. 2) is defined as:

$$\forall e_n, e_s \in \mathcal{E}_{\mathcal{NS}}, \forall e_w, e_e \in \mathcal{E}_{\mathcal{WE}}, \exists t \in \mathcal{T} \text{ s.t. } t = (e_n, e_s, e_w, e_e)$$

A complete tileset contains: $|\mathcal{T}| \geq |\mathcal{E}_{\mathcal{NS}}|^2 \cdot |\mathcal{E}_{\mathcal{WE}}|^2$.

2) Sub-Complete Tileset: The sub-complete tileset (Example as Fig. 3) is defined as:

$$\forall e_n, e_s \in \mathcal{E}_{\mathcal{NS}}, \exists t \in \mathcal{T} \text{ s.t. } e_n(t) = e_n \wedge e_s(t) = e_s$$
$$\forall e_w, e_e \in \mathcal{E}_{\mathcal{WE}}, \exists t \in \mathcal{T} \text{ s.t. } e_w(t) = e_w \wedge e_e(t) = e_e$$
$$\forall e_n \in \mathcal{E}_{\mathcal{NS}}, \forall e_w \in \mathcal{E}_{\mathcal{WE}}, \exists t \in \mathcal{T} \text{ s.t. } e_n(t) = e_n \wedge e_w(t) = e_w$$
$$\forall e_s \in \mathcal{E}_{\mathcal{NS}}, \forall e_e \in \mathcal{E}_{\mathcal{WE}}, \exists t \in \mathcal{T} \text{ s.t. } e_s(t) = e_s \wedge e_e(t) = e_e$$

The sub-complete tileset requires creating tiles for at least: $|\mathcal{T}| \geq \max\{|\mathcal{E}_{\mathcal{NS}}|^2, |\mathcal{E}_{\mathcal{WE}}|^2\}$. There are many combinations of sub-complete tilesets. If two tilesets $\mathcal{T}_1, \mathcal{T}_2$ have the same edge sets $\mathcal{E}_{\mathcal{NS}}$ and $\mathcal{E}_{\mathcal{WE}}$, $\mathcal{T}_1$ is a sub-complete tileset, $\mathcal{T}_2$ is a complete tileset, then $\mathcal{T}_1 \subseteq \mathcal{T}_2$.

The sub-complete tileset meets all the characteristics of the complete tileset, but under the edge set of the same size, the number of tiles required by the sub-complete tileset is much smaller than that of the complete tileset, making it a more practical option for game production.

B. Infinity

The infinity characteristic ensures algorithm will always return an accepted solution regardless of the range of the problem.

1) Definition: An *infinity accepted* tiling $f : \mathbf{G} \leftarrow \mathcal{T}$ is:

$$\forall_{m=1}^{\infty} \forall_{n=1}^{\infty} |g_{m,n}| = 1$$
$$\wedge \, e_n(m,n) = e_s^*(m-1,n) \wedge e_s(m,n) = e_n^*(m+1,n)$$
$$\wedge \, e_w(m,n) = e_s^*(m,n-1) \wedge e_e(m,n) = e_w^*(m,n+1)$$

2) Infinity of Complete Tileset: It is easy to prove that the complete tileset can satisfy the definition. For a cell $g_{m,n}$, the most strict constraint that a tile $t \in \mathcal{T}$ should satisfy the edge constraints propagate its four adjacent collapsed sub-grids, such that:

$$e_n(t) = e_s(m-1,n) \wedge e_s(t) = e_n(m+1,n) \wedge$$
$$e_w(t) = e_e(m,n-1) \wedge e_e(t) = e_w(m,n+1)$$

According to the definition of the complete tileset. We can always find at least one tile to satisfy such constraints.

3) Infinity of Sub-complete Tileset: To prove the infinity on the sub-complete tileset. We can use the diagonal generation process as mentioned in N-WFC. Suppose the infinity board $\mathbf{G}$ starts from the top-left corner $\mathbf{G}^{1,1}$, and the generation sequence of each tile follows the diagonal generation process. In this process, the most edge constraints a tile will encounter are when it is set with two edges already constrained by its upper and left adjacent tiles. For a cell $g_{m,n}$, we need to find a tile $t \in \mathcal{T}$, such that:

$$e_n(t) = e_s(m-1,n) \wedge e_w(t) = e_e(m,n-1)$$

According to one definition of the sub-complete tileset:

$$\forall e_n \in \mathcal{E}_{\mathcal{N}\mathcal{S}}, \forall e_w \in \mathcal{E}_{\mathcal{W}\mathcal{E}}, \exists t \in \mathcal{T} \text{ s.t. } e_n(t) = e_n \wedge e_w(t) = e_w$$

We can always find at least one tile that satisfies such constraints.

C. Aperiodicity

The aperiodicity characteristic ensures the algorithm generates non-repetitive content, meaning each generation has a different result, making the game more Rogue-like.

1) Definition: A tiling is called *periodic* with period $(a,b) \in \mathbb{Z}^2$ $f : \mathbf{G} \leftarrow \mathcal{T}$ if

$$\forall (m,n) \in \mathbb{Z}^2 : |g_{m,n}| = 1 \wedge g_{m,n} = g_{m+a,n+b}$$

f is called *aperiodicity accepted* if f is not periodic.

2) Aperiodicity of Complete and Sub-Complete Tileset: Suppose the generation process on $\mathbf{G}$ still uses diagonal generation, consider an arbitrary cell $g_{m,n}$ that has not been assigned a tile yet. By the edge constraints, this cell has two neighbors $g_{m,n-1}, g_{m-1,n}$ to the north and west that have already been assigned tiles. Note that if $m = 1$ or $n = 1$, the corresponding neighbor may not exist, but that is not a problem as we can assume that such neighbors are assigned an arbitrary tile that satisfies the missing constraint.

Either $\mathcal{T}$ is complete or sub-complete, according to the definition, there exists at least two tiles $t \in \mathcal{T}$ that satisfies the two edge constraints $e_n(t) = e_s(m-1,n), e_w(t) = e_e(m,n-1)$. Thus, at least two choices of tiles for $g_{m,n}$ do not repeat any previous configuration, which causes an independent random process. This argument applies to any uncollapsed cell, making the tiling process f using the complete or sub-complete tileset an aperiodic tiling.

D. Determinacy

The characteristic of *determinacy* is a context definition. It is defined based on combining the use of N-WFC and sub-complete tileset. While the fine-grained diagonal generation process may be used directly, the combination of N-WFC and sub-complete tilesets is more suitable for design purposes, as we will discuss in section V.

The N-WFC framework is well-suited for infinite content generation. In practice, *infinity* means generating scenes before the user reaches them. Marian Kleinebery [20], the creator of *infinite city*, mentioned that WFC has the problem that some places, although shown to the player, require backtracking and re-generation due to conflicts.

> *In my opinion, this limitation makes the WFC approach for infinite worlds unsuitable for commercial games.*

Determinacy is the characteristic that directly solve this problem. We prove that each sub-grid N-WFC has generated will never be backtracked during the whole generation process.

1) Definition: A N-WFC tiling $f : \mathbf{G} \leftarrow \mathcal{T}, \mathcal{T}$ is called *determinacy* if

$$\forall_{a=1}^{A} \forall_{b=1}^{B} f : \mathbf{G}^{a,b} \leftarrow \mathcal{T} \text{ is an accepted I-WFC result}$$

2) Determinacy of Sub-complete Tileset: For N-WFC, The strongest constraint is $\exists \mathbf{G}^{sub}, \forall m \in [1,C], |g_{m,1}^{sub}| = 1$, $\exists \mathbf{G}^{sub}, \forall n \in [1,C], |g_{1,n}^{sub}| = 1$. As we break $\mathbf{G}^{sub}$ into cells, all the pre-constraints are related to $e_n(t)$ and $e_w(t)$. As proved in section IV-B, the diagonal generation process always produces one solution for a sub-grid. Since I-WFC uses the backtracking strategy, it is guaranteed to return an accepted solution if one exists. Therefore, N-WFC with the sub-complete tileset is deterministic.

V. EXPERIMENT AND EVALUATION

A. Time Complexity of N-WFC and WFC in Sub-complete Tileset

We evaluated the sub-complete tileset's performance using the N-WFC framework and compared it with the traditional WFC algorithm. Both of them used the Minimum Remaining Value strategy for wave observation and Random Selection Strategy for wave collapse. In the exterior of the N-WFC, we used the diagonal generation process.

To evaluate the time complexity, we created a series of sub-complete tilesets with different numbers of edge sets. We assigned the value C a small constant for I-WFC and evaluated the time complexity for different size of grid matrix. For each size, the experiment was executed 100 times, and we calculated the mean and variance of the time.

Fig. 4. shows the result of the time cost experiment. We created 6 different sub-complete tilesets with different sizes of edge sets. For simplicity, we set the edge set $|\mathcal{E}_{\mathcal{N}\mathcal{S}}| = |\mathcal{E}_{\mathcal{W}\mathcal{E}}| = \{2,3,4,5,6,7\}$, and we assigned $C = 5$. We evaluated it through different board size $[(5 \times 9), (9 \times 17), (17 \times 33), (25 \times 49), (33 \times 75), (41 \times 81), (49 \times 97)]$
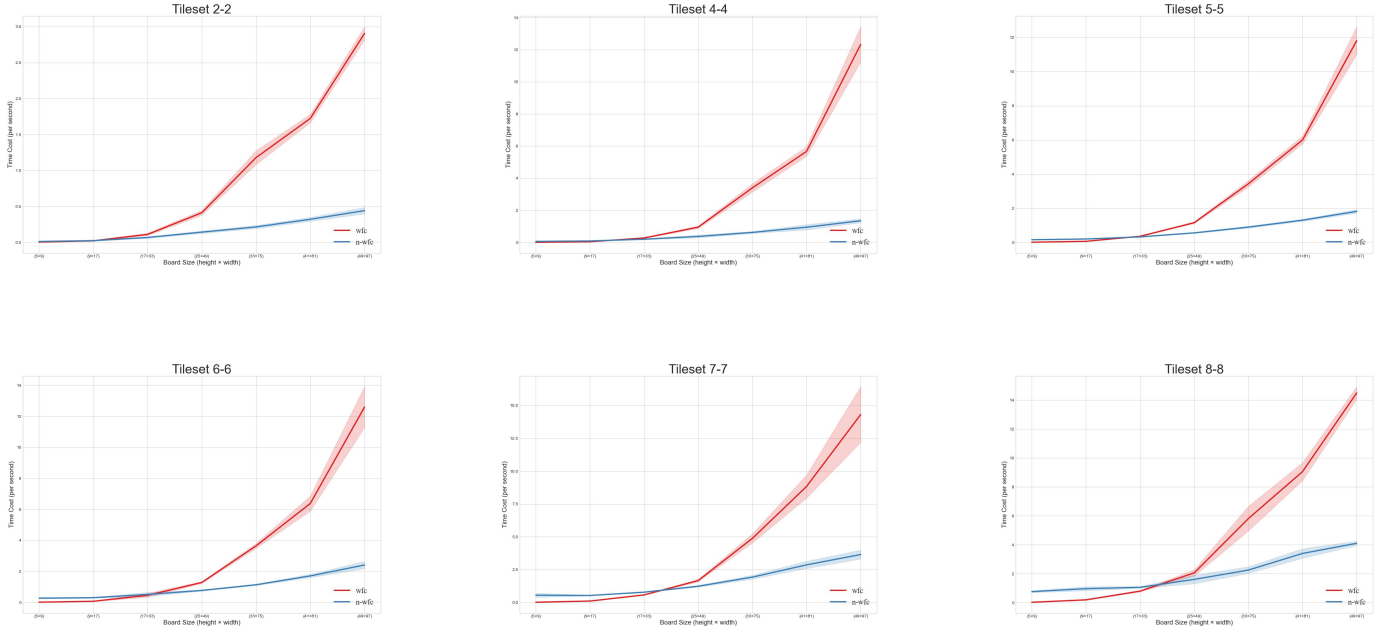
Fig. 4. Time cost comparison between WFC and N-WFC among different sub-complete tilesets.

The results show that the time complexity of WFC grows exponentially, while N-WFC grows in polynomial time complexity. For large-scale content generation, N-WFC only uses I-WFC to backtrack the fixed range of map tiles, outperforming traditional WFC.

B. Implementation Example: Carcassonne

As an example of deterministic large-scale or infinite content generation, we used the board game Carcassonne, which provides logically four edge types: e_0 for grass, e_1 for city, e_2 for path, and e_3 for stream. Specifically, in this game, $\mathcal{E}_{\mathcal{NS}} = \mathcal{E}_{\mathcal{WE}} = \{e_0, e_1, e_2, e_3\}$. We created a sub-complete tileset that satisfies the requirements as shown in Fig. 5-a. It consists of several tiles as shown in Table I. The sub-complete tileset we're using is a very small subset. In the actual implementation of the game, tiles can be either symmetry or rotated as needed. Multiple tiles with identical logical form are also accepted, providing more game combinations.

1) Large-Scale Content: To generate a large-scale scene, we set the size of the board to be generated by N-WFC (Fig. 5-b) and passed the constraints of the board's adjacent grid to the I-WFC generation process in diagonal order (Fig. 5-c). When all sub-grids are generated, each of them share one edge with another adjacent sub-grid, and are overlapped to form the final result (Fig. 5-d). Our algorithm can generate a map of approximately 50,000 tiles within 10 seconds.

2) Infinite Content: To generate infinite content, we modify the exterior generation process by taking the player as the origin and pre-generating the closest nine sub-grids of I-WFC with the player at the center (Fig. 6-b). When the player moves to a different position, N-WFC detects any ungenerated sub-grids in the nine adjacent sub-grids whose origin is the new

TABLE I
ONE EXAMPLE OF CARCASSONNE SUB-COMPLETE TILESET

tileset			
(e_0, e_0, e_0, e_0)	(e_2, e_0, e_2, e_0)	(e_0, e_0, e_1, e_1)	(e_2, e_2, e_0, e_0)
(e_0, e_1, e_0, e_1)	(e_2, e_1, e_2, e_1)	(e_0, e_0, e_2, e_2)	(e_2, e_2, e_1, e_1)
(e_0, e_2, e_0, e_2)	(e_2, e_2, e_2, e_2)	(e_0, e_0, e_3, e_3)	(e_2, e_2, e_3, e_3)
(e_0, e_3, e_0, e_3)	(e_2, e_3, e_2, e_3)	(e_1, e_1, e_0, e_0)	(e_3, e_3, e_0, e_0)
(e_1, e_0, e_1, e_0)	(e_3, e_0, e_3, e_0)	(e_1, e_1, e_2, e_2)	(e_3, e_3, e_1, e_1)
(e_1, e_1, e_1, e_1)	(e_3, e_1, e_3, e_1)	(e_1, e_1, e_3, e_3)	(e_3, e_3, e_2, e_2)
(e_1, e_2, e_1, e_2)	(e_3, e_2, e_3, e_2)	(e_1, e_3, e_1, e_3)	(e_3, e_3, e_3, e_3)

position and continuously generates new sub-grids using I-WFC (Fig. 6-c,d).

Using the sub-complete tileset, I-WFC is guaranteed to return an accepted solution given two adjacent row and column constraints propagated from the pre-generated sub-grids. The deterministic characteristic of the generation process ensures that all the ranges that players have visited will not be backtracked. Since I-WFC is generated on a smaller grid, it completes the generation process in undetectable time, providing the player with an infinite scene.

C. Device Configuration

The whole experiment was executed on a computer with Microsoft Windows 11 Pro OS, Intel(R) Core(TM) 17-9750H CPU @ 2.60GHz, and 16GB RAM.

VI. DISCUSSION

A. Design Proprieties and Weight Brush System

In Section IV, we discussed the logical forms of sub-complete tilesets. However, tiles with identical logical forms can have different design properties, such as artistic expressions. We propose a *Weight Brush System* that integrates design
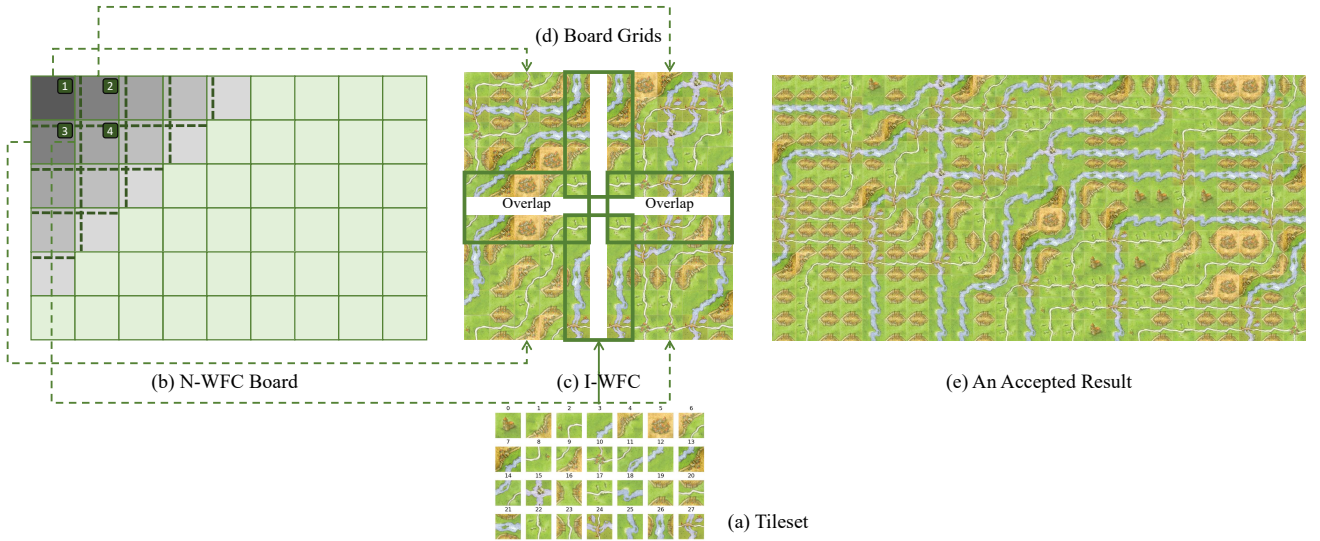
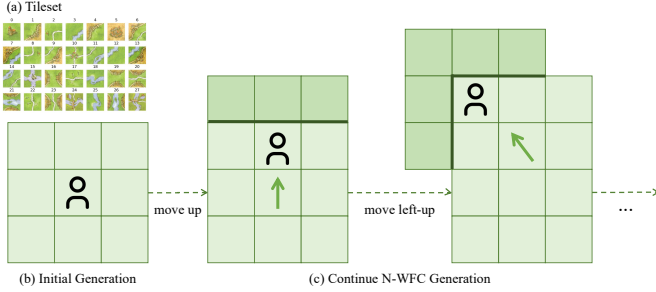Fig. 5. Large-scale game implementation with N-WFC and sub-complete tileset

Fig. 6. Infinite game implementation with N-WFC and sub-complete tileset

Fig. 7. Logically identical tiles have different design properties (Tile (a) and (e) are Carcassonne original tiles, while DALL·E generates the others)

properties with the WFC algorithm, making the generation process more flexible. These design properties can be controlled during the generation process. As Fig. 7 shows, (a)-(d) are identical tiles $t = (e_0, e_0, e_0, e_0)$ but with different artistic expression. In the design, we can think of tile (a) as an ordinary house and tile (d) as a luxury house. This is the same as tile (e) is more luxurious than tile (h). Tag *luxurious* is a design property that can vary and share among different tiles.

The Weight Brush System consists of different weighted brushes, each representing tiles with a specific design property tag, such as buildings, landscapes, or quest points. Game designers can customize the weights of the brushes and use them to draw rough N-WFC generated scenes. The weights affect the selection of a cell to collapse and the choice of which tile to collapse and can change the heuristic policy of the traditional WFC algorithm.

We argue that the weighted brush system is better than the diagonal generation process for generating sub-grids. If the I-WFC uses diagonal generation process to collapses cells in a fixed order. It prevents design-associated algorithms with selection strategies from being applied. In contrast, the weighted brush system can be integrated with such algorithms. It emphasizes game designers' priorities and produces maps
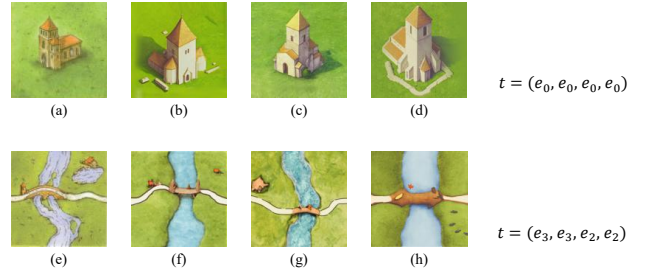
with better design.

B. Expand N-WFC and Sub-complete Tileset to 3D

We have focused on implementation strategies for the 2D case, but the approach can be easily extended to 3D. A 3D tile (voxel) consists of six edges (faces), denoted by $t = (e_n, e_s, e_w, e_e, e_u, e_d)$, which belong to three different face sets $\mathcal{E}_{\mathcal{NS}}, \mathcal{E}_{\mathcal{WE}}, \mathcal{E}_{\mathcal{UD}}$. N-WFC operates on a grid matrix $\mathbf{G} \subseteq \mathbb{Z}^3$ consisting of $M \times N \times W$ tiles. In the 3D case, N-WFC uses I-WFC with fixed $C \times C \times C$ sub-grid. The time complexity of N-WFC in 3D is $O(\frac{M \times N \times W}{C^3} d^{C^3} + (M \times N \times W)C^3 d^3)$. We can also add five new constraints to the definition of the sub-complete tileset.

$$\forall e_u, e_d \in \mathcal{E}_{\mathcal{UD}}, \exists t \in \mathcal{T} \text{ s.t. } e_u(t) = e_u \wedge e_d(t) = e_d$$
$$\forall e_n \in \mathcal{E}_{\mathcal{NS}}, \forall e_u \in \mathcal{E}_{\mathcal{UD}}, \exists t \in \mathcal{T} \text{ s.t. } e_n(t) = e_n \wedge e_u(t) = e_u$$
$$\forall e_s \in \mathcal{E}_{\mathcal{NS}}, \forall e_d \in \mathcal{E}_{\mathcal{UD}}, \exists t \in \mathcal{T} \text{ s.t. } e_s(t) = e_s \wedge e_d(t) = e_d$$
$$\forall e_e \in \mathcal{E}_{\mathcal{WE}}, \forall e_u \in \mathcal{E}_{\mathcal{UD}}, \exists t \in \mathcal{T} \text{ s.t. } e_e(t) = e_e \wedge e_u(t) = e_u$$
$$\forall e_w \in \mathcal{E}_{\mathcal{WE}}, \forall e_d \in \mathcal{E}_{\mathcal{UD}}, \exists t \in \mathcal{T} \text{ s.t. } e_w(t) = e_w \wedge e_d(t) = e_d$$

In the case of 3D, the sub-complete tileset also satisfies the three characteristics. The exterior of the N-WFC can use

different generation methods to generate scenes that match the design expectations.

C. Limitation and Future Work

While we have demonstrated that the N-WFC of the simple tiled model using sub-complete tilesets can generate deterministic infinite content in a short time, the overlapping model has the advantage of automatically extracting the constraint relationship from an existing picture or model. Still, it currently fails to satisfy the sub-complete constraint. It creates large edge sets but is unable to extract enough tiles that meet the definition of the sub-complete tileset. This results in a lot of conflicts during N-WFC generation. Future work may consider modifying the overlapping model and implementing post-processing to meet the demands of sub-complete tilesets.

VII. CONCLUSION

In this paper, we introduced Nested WFC (N-WFC), an optimization strategy for the WFC algorithm. N-WFC nests an interior WFC (I-WFC) into a large exterior generation process while maintaining the edge constraints, reducing exponential complexity to polynomial complexity. We proposed the concept of complete and sub-complete tilesets to maintain edge constraints during the N-WFC generation process and proved that under such tilesets, N-WFC can generate deterministic, aperiodic, and infinite content. We demonstrated through experiments that the combination of N-WFC and sub-complete tileset meets time complexity expectations. We provided an example of N-WFC's application in game development, specifically the Carcassonne game. We also proposed a weighted brush system to better generate scenes conforming to the design. Our work addresses the backtracking and conflicting shortcomings of WFC for large-scale and unlimited content generation meets the needs of game designers and is expandable and robust.

REFERENCES

[1] M. Gumin, "Wave function collapse algorithm," Sep 2016. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse

[2] F. Games, "Caves of qud," 2014. [Online]. Available: https://www.cavesofqud.com/

[3] [Online]. Available: https://www.badnorth.com

[4] O. Stålberg, "Townscraper," Aug 2021. [Online]. Available: https://store.steampowered.com/app/1291340/_/

[5] R. Bailly and G. Levieux, "Genetic-wfc: Extending wave function collapse with genetic search," *IEEE Transactions on Games*, 2022.

[6] I. Karth and A. M. Smith, "Wavefunctioncollapse: content generation via constraint solving and machine learning," *IEEE Transactions on Games*, vol. 14, no. 3, pp. 364–376, 2021.

[7] Wikipedia, Jan 2023, page Version ID: 1135642137. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Roguelike

[8] Epyx, "Rogue," Jun 1985. [Online]. Available: https://store.steampowered.com/app/1443430/Rogue/

[9] H. Wolverson, "Proguelike tutorial - in rust," https://github.com/amethyst/rustrogueliketutorial, 2022.

[10] J. R. Baron, "Procedural dungeon generation analysis and adaptation," in *Proceedings of the SouthEast Conference*, 2017, pp. 168–171.

[11] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular automata for real-time generation of infinite cave levels," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Monterey California: ACM, Jun 2010, p. 1–4. [Online]. Available: https://dl.acm.org/doi/10.1145/1814256.1814266

[12] A. Koesnaedi and W. Istiono, "Implementation drunkard's walk algorithm to generate random level in roguelike games," *International Journal of Multidisciplinary Research and Publications*, vol. 5, no. 2, pp. 97–103, 2022.

[13] F. Aurenhammer and R. Klein, "Voronoi diagrams," *Handbook of computational geometry*, vol. 5, no. 10, pp. 201–290, 2000.

[14] J. C. Hart, "Perlin noise pixel shaders," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 2001, pp. 87–94.

[15] M. Gumin, "Markovjunior, a probabilistic programming language based on pattern matching and constraint propagation," Jun 2022. [Online]. Available: https://github.com/mxgmn/MarkovJunior

[16] O. E. L. Team, A. Stooke, A. Mahajan, C. Barros, C. Deck, J. Bauer, J. Sygnowski, M. Trebacz, M. Jaderberg, M. Mathieu *et al.*, "Open-ended learning leads to generally capable agents," *arXiv preprint arXiv:2107.12808*, 2021.

[17] P. Merrell, "Comparing model synthesis and wave function collapse," Jul 2021. [Online]. Available: https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf

[18] I. Karth and A. M. Smith, "Wavefunctioncollapse: Content generation via constraint solving and machine learning," *IEEE Transactions on Games*, vol. 14, no. 3, p. 364–376, Sep 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9421370/

[19] H. Kim, T. Hahn, S. Kim, and S. Kang, "Graph based wave function collapse algorithm for procedural content generation in games," *IEICE TRANSACTIONS on Information and Systems*, vol. 103, no. 8, pp. 1901–1910, 2020.

[20] M. Kleineberg. [Online]. Available: https://marian42.itch.io/wfc

[21] F. Mourato, M. P. Dos Santos, and F. Birra, "Automatic level generation for platform videogames using genetic algorithms," in *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. Lisbon Portugal: ACM, Nov 2011, p. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/2071423.2071433

[22] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "Pcgrl: Procedural content generation via reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, p. 95–101, Oct 2020. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/7416

[23] G. Smith, *Techniques for AI-Driven Experience Management in Interactive Narratives*, 0th ed. A K Peters/CRC Press, Apr 2015, p. 552–563. [Online]. Available: https://www.taylorfrancis.com/books/9781482254808/chapters/10.1201/b18373-48

[24] O. Stålberg, "Wave - by oskar stålberg." [Online]. Available: https://oskarstalberg.com/game/wave/wave.html