Analyzing Procedural Terrain Generation in Games from a Constraint Programming Perspective

André Felipe de Almeida Pontes[1], **Gabriel Pimentel Dolzan**[1], **Guilherme Alex Derenievicz**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)

{afap19,gpd20,guilherme}@inf.ufpr.br

Abstract. *Procedural generation of two-dimensional terrains serves as a fundamental strategy in the development of many digital games. In 2016, the algorithm Wave Function Collapse (WFC) stood out as an effective technique for generating small-scale terrains. Aiming to address larger instances, a variant called Nested WFC (N-WFC) was proposed in 2023. This paper presents an analysis of both WFC and N-WFC from the perspective of the Constraint Programming paradigm, in which the relations that define the terrain validity are modeled by constraints. An experimental analysis, conducted in Unreal Engine and in C++ with both algorithms indicates that for specific tilesets the directed arc consistency is sufficient to generate the terrain without performing backtracking.*

Resumo. *A geração procedural de terrenos bidimensionais se apresenta como uma estratégia fundamental no desenvolvimento de muitos jogos digitais. Em 2016, o algortimo Wave Function Collapse (WFC) se destacou como uma técnica eficaz para a geração de terrenos de pequeno porte. Com o objetivo de abordar instâncias maiores, uma variante denominada Nested WFC (N-WFC) foi proposta em 2023. Este artigo apresenta uma análise do WFC e do N-WFC pela perspectiva do paradigma de Programação por Restrições, no qual as relações que definem a validade dos terrenos são modeladas por meio de restrições. Uma análise experimental, conduzida na Unreal Engine e em C++, com ambos algoritmos, indica que para conjuntos específicos de tiles a consistência de arco direcionada é suficiente para gerar o terreno sem efetuar backtracking.*

1. Introdução

Na área da Inteligência Artificial (IA), os algoritmos de geração procedural (PCG, do inglês *Procedural Content Generation*) são utilizados para reduzir tempo e custo de desenvolvimento de jogos, criando recursos dinamicamente e tornando a experiência mais imersiva e personalizada para cada usuário [Mehta 2025]. Estudos de mercado também indicam a importância do segmento de PCGs: em 2023, esse segmento detinha mais de 30% da fatia do mercado de IA em jogos [Market.US 2024].

Em 2016, o algoritmo *Wave Function Collapse* (WFC) se destacou como uma técnica simples e eficaz para geração procedural de terrenos bidimensionais, sendo utilizado em diversos jogos comerciais como *Bad North*, *Caves of Qud* e *Townscaper* [Gumin 2016]. Devido à sua capacidade de gerar padrões complexos e variados a partir de regras simples ou pré-definidas, tornou-se popular entre desenvolvedores de jogos.

Neste contexto, o *Nested Wave Function Collapse* (N-WFC) foi proposto como uma variante do WFC original [Nie et al. 2023]. O algoritmo executa múltiplas instâncias pequenas do WFC em uma sequência de geração denominada **diagonalização** que reduz conflitos e acelera o processo de geração de terreno. Em particular, para conjuntos específicos de *tiles*, denominados *tileset* completo e sub-completo, o N-WFC garante a geração de conteúdo infinito, aperiódico e determinístico, sem a necessidade de *backtracking* [Nie et al. 2023].

Neste trabalho, os algoritmos WFC e N-WFC são analisados pela perspectiva do paradigma de Programação por Restrições. A geração procedural de terrenos pode ser definida como um Problema de Satisfação de Restrições (CSP, do inglês *Constraint Satisfaction Problem*) no qual a grade do terreno a ser construído é representada por um conjunto de variáveis, o *tileset* é mapeado no conjunto de domínios das variáveis, e as relações de validade entre as *tiles* do terreno são representadas por restrições binárias entre variáveis. Nesse contexto, o objetivo deste artigo é relacionar o WFC e sua variante a conceitos bem estabelecidos na área de Programação por Restrições, como a consistência direcionada e o grafo de restrições.

Para fins de validação, duas análises experimentais são conduzidas, uma delas na *Unreal Engine*, com otimizações e tecnologias modernas que permitem criar jogos de alta qualidade, e outra em C++, com bibliotecas próprias. Os resultados mostram que a escolha do algoritmo pode depender fortemente da plataforma. De modo geral, o estudo conduzido sugere que os algoritmos de geração procedural de terrenos representam um tipo específico de CSPs nos quais é possível definir os domínios das variáveis de modo a satisfazer algum nível de consistência e, assim, possibilitar métodos de busca mais eficientes. O estudo busca contribuir para o avanço da pesquisa em geração procedural de conteúdos, através da análise de dados sobre a aplicabilidade, eficiência e utilização das técnicas abordadas para jogos e simulações de ambientes virtuais.

O restante deste artigo está organizado da seguinte forma: na Seção 2 são apresentados os fundamentos de Programação por Restrições e de geração procedural de terrenos; a Seção 3 apresenta uma análise do WFC e do N-WFC, relacionando-os com conceitos de CSPs; a Seção 4 apresenta os resultados experimentos e a Seção 5 conclui o artigo.

2. Fundamentos

2.1. Programação por Restrições

A Programação por Restrições é um paradigma declarativo de problemas combinatórios por meio de restrições. Esse paradigma envolve a modelagem de problemas em termos de variáveis e restrições, que definem o conjunto de soluções possíveis. Neste caso, obtêm-se um Problema de Satisfação de Restrições (CSP), cujo objetivo é encontrar uma valoração às variáveis de modo a satisfazer todas as restrições simultâneamente [Rossi et al. 2006].

Formalmente, uma instância do CSP é uma tripla (X, D, C), onde $X = \{X_1, X_2, \ldots, X_n\}$ é um conjunto de **variáveis**; $D = \{D_1, D_2, \ldots, D_n\}$ é o conjunto dos **domínios** das variáveis, onde cada domínio representa o conjunto de valores que a variável pode assumir; e $C = \{C_1, C_2, \ldots, C_m\}$ é um conjunto de **restrições**, que especificam quais valores são permitidos para as variáveis. Assim, para toda restrição C_j envolvendo as variáveis $S_j = \{X_{j_1}, X_{j_2}, \ldots, X_{j_k}\}$, a tupla de valores $(v_{j_1}, v_{j_2}, \ldots, v_{j_k})$

atribuída às variáveis deve pertencer ao conjunto permitido definido pela restrição: $(v_{j_1}, v_{j_2}, \ldots, v_{j_k}) \in C_j$. Uma forma de representar as restrições é, portanto, enumerar todas as combinações válidas para as variáveis às quais a restrição se refere.

Um outro aspecto do paradigma de Programação por Restrições é a definição de estratégias inteligentes de busca por soluções, geralmente combinando *backtracking* com técnicas de propagação de restrições, como a consistência de arco e consistências direcionadas [Rossi et al. 2006]. A busca por ***backtracking*** é uma técnica de busca em profundidade para CSP [Russell and Norvig 2010]. A cada nível, escolhe-se uma variável não atribuída e um valor do seu domínio que respeite as restrições. Se tal valor não existir, é necessário desfazer a última atribuição e tentar outro valor. Esse processo ocorre continuamente até que uma atribuição completa, consistentes com as restrições, seja encontrada ou todas as opções sejam exploradas. De acordo com [Russell and Norvig 2010], algumas heurísticas comuns para escolhas de variáveis são:

- ***Minimum Remaining Values* (MRV):** escolher a variável com menor número de valores remanescentes. Essa heurística é utilizada no WFC padrão [Gumin 2016];
- ***Degree Heuristic*:** critério de desempate escolhendo a variável envolvida em mais restrições com outras não atribuídas;
- ***Least Constraining Value*:** testar primeiro o valor que elimina menos opções nas variáveis que compartilham alguma restrição;

A busca por *backtracking* é geralmente acompanhada pela aplicação de técnicas de **propagação de restrições**, que utilizam as estruturas das restrições para remover valores inconsistentes dos domínios das variáveis ou gerar novas restrições que estão explícitas no problema. A técnica de propagação mais conhecida é chamada **consistência de arco**, que atua sobre restrições binárias (i.e, restrições sobre 2 variáveis) e garante que para todo valor no domínio de uma variável existe um valor viável no domínio da outra variável com quem compartilha uma restrição. A implementação mais usada dessa consistência chama-se AC-3 [Mackworth 1977], descrita no Algoritmo 1.

Algoritmo 1: AC-3

Entrada: CSP (X, D, C)
Saída: Domínios D com consistência de arco
$Fila \leftarrow C$;
Enquanto $Fila \neq vazia$ **faça**
  $C_j \leftarrow$ remover uma restrição da $Fila$;
  seja $S_j = \{X_{j_1}, X_{j_2}\}$ as variáveis da restrição C_j;
  Para cada $a \in D_{j_1}$ **faça**
    Se *não existe* $b \in D_{j_2}$ *tal que* $(a, b) \in C_j$ **então**
      remover a de D_{j_1};
  Se *removeu algum valor de* D_{j_1} **então**
    Para cada *restrição* C_k *sobre* X_{j_1}, $k \neq j$ **faça**
      adicionar C_k na $Fila$;
Retorne D;

Em cada passo, garante que todo valor em D_{j_1} tenha ao menos um suporte em D_{j_2}, removendo valores inconsistentes. Se D_{j_1} for reduzido, volta a verificar todas as restrições

sobre X_{j_1}, até estabilizar todos os domínios. A complexidade no pior caso é $O(cd^3)$, onde c é o número de restrições e d o tamanho máximo de domínio [Russell and Norvig 2010].

2.2. Geração Procedural de Terrenos em Jogos

A geração procedural de conteúdos consiste na definição automática de elementos do jogo que não estão completamente especificados durante o seu desenvolvimento, podendo se referir a níveis, mapas, regras, texturas, itens, missões, músicas, armas, veículos, personagens, histórias, entre outros [Shaker et al. 2016]. Por sua vez, a geração procedural de terrenos é um sub-problema da geração procedural de conteúdos e refere-se a criação automática de terrenos e texturas [Rose and Bakaoukas 2016].

Segundo [Togelius et al. 2011], existem muitos argumentos para a utilização desse tipo de abordagem para criação de conteúdos para jogos. Um deles é a redução da memória utilizada, mantendo o mundo "comprimido" até que seja necessário expandí-lo. Outro argumento para a utilização desses algoritmos é a facilidade de não ter que criar cada aspecto desses conteúdos manualmente, o que poderia aumentar significativamente os recursos e o tempo gasto na produção desses *softwares*. Além disso, jogos podem utilizar tais algoritmos como sua mecânica principal, possibilitando novos estilos de jogos onde a re-jogabilidade é maior.

2.2.1. WFC

O algoritmo WFC, para geração procedural de terrenos, foi proposto por [Gumin 2016] inspirado em ideias da mecânica quântica. A entrada do problema é uma matriz (*grid*) de tamanho $n \times n$ e um conjunto de padrões (*tileset*) pré-definido, com regras bem definidas de adjacência em cada direção (norte, sul, leste, oeste). Um exemplo de *tileset* e de um terreno de tamanho 10×10 é mostrado na Figura 1. As regras de validade, neste caso, é que bordas devem coincidir em estradas ou campo.

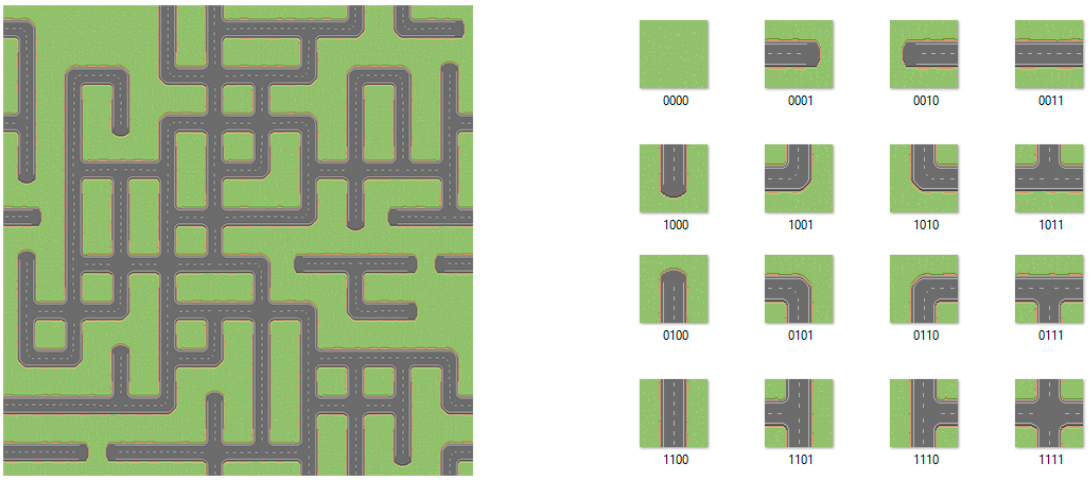

Figura 1. Exemplo de uma imagem 10x10 gerada ao lado do *tileset*, obtido de [Kubi 2021].

De modo geral, o WFC consiste nos seguintes passos:

1. **inicialização:** todas as células da matriz começam não colapsadas (isto é, em um estado de sobreposição em todo o *tileset*);
2. **seleção de célula:** escolher a célula não colapsada com menor entropia, isto é, com o menor número efetivo de padrões possíveis, de acordo com as regras das bordas;
3. **colapso:** sortear um dos possíveis padrões para ser colocado naquela posição.
4. **propagação:** para cada vizinho da célula colapsada, eliminar todos os padrões cuja região sobreposta não coincide com a do padrão recém-selecionado. Essa etapa continua até que a matriz esteja estabilizada;
5. ***backtracking*:** caso uma célula fique vazia, isto é, sem opções de padrões viáveis, é necessário realizar *backtracking*, desfazendo a etapa de colapso anterior e tentando outros padrões disponíveis. Caso contrário, se ainda houver células não colapsadas, voltar ao passo 2.

No pior caso, o WFC demanda tempo de processamento exponencial no tamanho da instância.

2.2.2. N-WFC

O N-WFC é uma variante do WFC, proposta por [Nie et al. 2023], na qual a matriz é dividida em *subgrids* de tamanho fixo que se sobrepõem e o WFC original é executado separadamente em cada subgrid. Em um laço externo as subgrids são percorridas em camadas diagonais. A sobreposição ocorre apenas na primeira coluna e na primeira linha de cada subgrid, exigindo que haja coincidência com as escolhas feitas na camada diagonal anterior. Os seguintes passos detalham o algoritmo:

1. **inicialização:** fragmentar a matriz em subgrids de tamanho $C \times C$ que se sobrepõem na primeira linha e coluna, e separá-los em camadas diagonais, cada camada iniciando no lado superior direito e terminando no lado inferior esquerdo. Começar com a camada mais à esquerda;
2. **seleção de célula:** escolher o próximo subgrid $G_{a,b}$ da camada diagonal corrente ou passar para a próxima camada;
3. **ajuste de restrições pré-estabelecidas:** colapsar previamente a primeira linha de $G_{a,b}$ com a última linha de $G_{a-1,b}$ e a primeira coluna de $G_{a,b}$ com a última coluna de $G_{a,b-1}$, se existirem;
4. **I-WFC:** executar o WFC em $G_{a,b}$ para obter um subgrid aceito;
5. **sobreposição:** inserir $G_{a,b}$ na matriz, sobrepondo as bordas pré-colapsadas e garantindo consistência entre subgrids adjacentes;
6. ***backtracking*:** caso o I-WFC não encontre uma solução para o subgrid, é necessário realizar *backtracking*, desfazendo a etapa do o I-WFC anterior e tentando outros padrões disponíveis. Caso contrário, se ainda houver células não colapsadas, voltar ao passo 2.

No artigo, [Nie et al. 2023] afirmam que a complexidade de tempo do N-WFC é polinomial se o *backtracking* do passo 6 nunca ocorrer, pois o I-WFC (passo 4), apesar de NP-Difícil no caso geral, tem tamanho limitado em cada subgrid ($C \times C$). Os autores ainda afirmam que para *tilesets* **completos** ou **sub-completos**, o tempo polinomial é garantido. Tais *tilesets* garantem a existência de todas as permutações possíveis de bordas em cada

lado dos padrões, evitando casos em que o *backtracking* seria necessário para corrigir inconsistências.

- ***tileset* completo:** é um conjunto que possui todas as combinações possíveis de bordas em cada uma das 4 direções (norte, sul, leste e oeste), requerendo uma grande quantidade de *tiles*.
- ***tileset* sub-completo:** é um conjunto que contém apenas as combinações mínimas necessárias, isto é, devem existir combinações entre as direções N/S, L/O, N/O e S/L, de modo que permite a geração diagonal do N-WFC. Isso garante que apesar de possuir uma quantidade bem menor de *tiles*, sempre será possível montar um grid completo.

3. Análise dos Algoritmos

Apesar do WFC ter sido inspirado por ideias da mecânica quântica [Gumin 2016], uma análise do algoritmo conduzida por [Merrell 2021] mostra que o WFC é basicamente um método para resolver CSP. De fato, a geração procedural de terrenos pode ser definida como um CSP no qual as variáveis representam as posições da matriz do terreno a ser preenchida, os domínios são compostos pelo *tileset* e as restrições modelam as relações de validade entre posições vizinhas da matriz. Neste cenário, o WFC mostra-se uma busca com *backtracking* padrão, que faz uso do AC-3 para propagar restrições, e da heurística MRV para escolha de variáveis (o equivalente a escolher a célula com "menor entropia").

A análise apresentada neste artigo se propõe a estender a relação entre o WFC e CSP considerando outros conceitos, provendo um novo entendimento sobre a variante N-WFC. A estrutura de uma instância CSP pode ser representada por um hipergrafo no qual vértices representam variáveis e hiper-arcos representam as restrições sobre as variáveis. O hipergrafo da esquerda na Figura 2 representa a estrutura geral de 4 células vizinhas em terreno bidimensional. Cada triângulo representa uma borda de uma célula (e.g, as variáveis A_N, A_L, A_S e A_O representam as bordas Norte, Leste, Sul e Oeste da célula A, respectivamente). Entre as 4 variáveis de uma célula existe uma restrição que garante que a escolha de valores para as 4 bordas constituam um *tile* válido. Se o *tileset* for completo, qualquer escolha é válida, isto é, qualquer atribuição de valores a essas variáveis satisfazem a restrição. No caso geral, no entanto, isso pode não ser satisfeito. Por exemplo, se o *tileset* apresentado na Figura 1 não tiver o *tile* 0000, então as atribuições de "grama" para as 4 bordas da célula não respeita a restriçao, pois não há *tile* correspondente. Entre diferentes células (e.g, A e B da Figura 2) há uma restrição de igualdade forçando a continuidade da borda no terreno.

Uma outra forma de representar uma instância CSP é através do **dual**: as restrições da instância original passam a ser as variáveis no CSP dual, sendo o domínio dessas variáveis conjuntos de tuplas (as combinações válidas na instância original). As variáveis duais devem respeitar restrições binárias de igualdade quando representam restrições do CSP original sobre as mesmas variáveis, portanto, a estrutura do dual pode ser sempre representada por um grafo simples. Na Figura 2, lado direito, é apresentado o grafo dual de uma matriz 3×3. A cada variável deve ser atribuída uma tupla de 4 valores (N,L,S,O) que respeitam a respectiva restrições do CSP original, isto é, devem constituir um *tile* válido. Entre variáveis, existe uma restrição de igualde entre bordas específicas, herdadas do problema original.

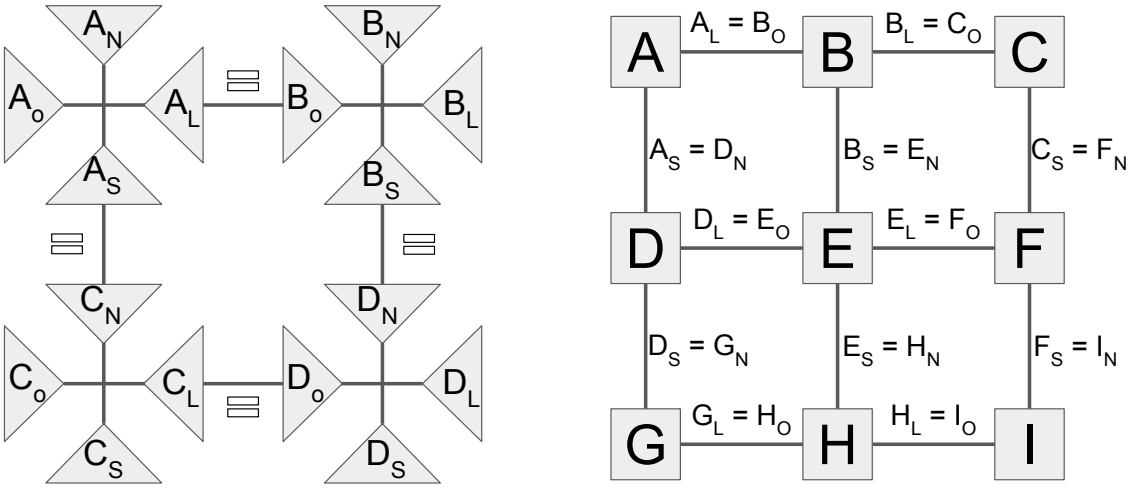

Figura 2. Hipergrafo de restrições de uma instância primal 2×2 (Esq.) e o grafo de restrições de uma instância dual 3×3 (Dir.).

A estrutura do CSP é relevante, pois identifica a complexidade do algoritmo de propagação de restrições necessário para resolver a instância sem efetuar *backtracking* [Freuder 1982]. Em particular, sabe-se que se o grafo de restrições for acíclico, o AC-3 é suficiente para resolver o problema, independente de quais forem os domínios das variáveis (*tileset*). No contexto da geração de mapas bidimensionais, a estrutura, no formato de grade, apresenta ciclos. Porém, é possível ordenar as variáveis de modo que cada variável tenha restrição com no máximo 2 variáveis que a antecedem na ordenação: basta seguir a ordem por camadas diagonais, como proposto no N-WFC, ou, mais simplificadamente, por linhas da matriz. No exemplo, da Figura 2, a ordenação $(A, B, C, D, E, F, G, H, I)$ respeita essa propriedade.

Este parâmetro é chamado de **largura** e identifica a quantidade necessária de consistência que precisa ser satisfeita pelos domínios das variáveis para garantir uma busca sem retrocesso [Freuder 1982]. No caso da estrutura em grade, com largura 2, a consistência de caminho é desejada: dado uma variável qualquer X, para quaisquer valores nas 2 variáveis que a antecedem na ordenação, existe um valor para X que respeita ambas as restrições. No caso da estrutura de terrenos, e seguindo a ordenação por linhas, isso significa que para toda borda Leste e toda borda Sul deve existir um *tile* cujas bordas Oeste e Norte encaixem na respectiva célula. A definição de *tileset* sub-completo, apresentado em [Nie et al. 2023], contempla essa propriedade.

Dessa forma, conclui-se que o fato do algoritmo N-WFC não efetuar *backtracking* e, portanto, ter complexidade polinomial, para *tilesets* sub-completos, advém do fato que esses *tilesets* constituem domínios que satisfazem a consistência de caminho no CSP correspondente. Além disso, entende-se que basta propagar as restrições de forma direcionada para que se obtenha uma solução, sem a necessidade de usar o I-WFC em subgrids da matriz. O Algoritmo 2 formaliza essa proposta. Para cada elemento da matriz,

propaga-se o *tile* escolhido aos seus vizinhos de baixo e da direita. O algoritmo demanda tempo polinomial no tamanho da instância e, assim como o I-WFC, resolve o problema para *tilesets* sub-completos.

Algoritmo 2: DAC

Entrada: CSP (X, D, C) representando um mapa $N \times N$
Saída: Solução *aceita*
Para $i = 1..N$ **faça**
  Para $j = 1..N$ **faça**
    escolher um *tile* t_1 em $D_{i,j}$ e atribuir a $X_{i,j}$
    Para cada tile t_2 *em* $D_{i,j+1}$ **faça**
      Se $t_2.O \neq t_1.L$ **então**
        remover *tile* t_2 de $D_{i,j+1}$;
    Para cada tile t_2 *em* $D_{i+1,j}$ **faça**
      Se $t_2.N \neq t_1.S$ **então**
        remover *tile* t_2 de $D_{i+1,j}$;

4. Resultados Experimentais

Os algoritmos WFC, N-WFC e DAC foram implementados em dois ambientes diferentes: *Unreal Engine*, e em C++ com a biblioteca `stb` [Barrett 2017] para manipular imagens (e.g. o mapa apresentado da Figura 1). Os experimentos com a *Unreal Engine* foram conduzidos em uma máquina AMD Ryzen 5600x, 4.6 GHz, com 32GB de memória RAM, executando Windows 11 v. 24H2 e Unreal 5.4, enquanto os experimentos em C++ foram conduzidos em uma máquina Intel i7-13700K, 3.40 GHz, com 32GB de memória RAM, executando Windows 11 v. 24H2. Todas as instâncias executadas possuem como *tileset* o apresentado na Figura 1.

A *Unreal Engine* é uma das engines mais versáteis e poderosas para o desenvolvimento de jogos e aplicações 3D em tempo real. Ela permite programar em C++ e, por meio de *Blueprints*, criar lógica de forma visual ("arrastar e soltar"). Além disso, dispõe de ferramentas avançadas para animação, física, áudio e iluminação. A escolha da *Unreal Engine* vem pela sua integração com C++, que possibilita alto desempenho, escalabilidade e controle de baixo nivel, além de suas ferramentas, que proporcionam um trabalho mais ágil. Para a implementação dos algoritmos foram utilizadas estruturas de dados do C++ da *Unreal*, como `TTuple`, `TArray` e `TMap`.

A Figura 3 apresenta resultados de tempo de processamento para os algoritmos implementados em C++. É possível perceber que todas as variantes são mais rápidas que o WFC original. Nota-se que foi possível tornar o WFC original mais rápido que o N-WFC, apenas fazendo a etapa de colapsar as células na mesma ordem que o N-WFC faz, isto é, utilizando a diagonalização (percorrendo todas as anti-diagonais), chamado no gráfico de DIAGONAL. Quanto ao DAC e a sua variante diagonal, nota-se que são mais rápidos que o WFC Diagonal, pois fazem somente 2 propagações por colapso de célula, ou seja, não utilizam AC-3 completo. Dessa forma, o algoritmo DAC proposto se mostrou o mais eficaz em termos de tempo de processamento em um *tileset* sub-completo.

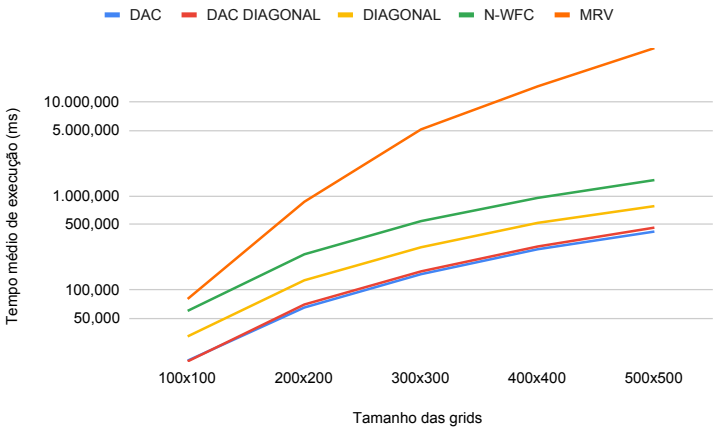

Figura 3. Tempo médio de execução (em ms) com base no tamanho da matriz, para todos os algoritmos implementados em C++.

O gráfico da Figura 4 apresenta os resultados para diferentes configurações do N-WFC, variando o tamnhos das subgrids. De modo geral, nota-se que o método fica mais rápido com $C = 5$, obtendo um bom compromisso entre a quantidade de passos do AC-3 e da sobreposição das subgrids.

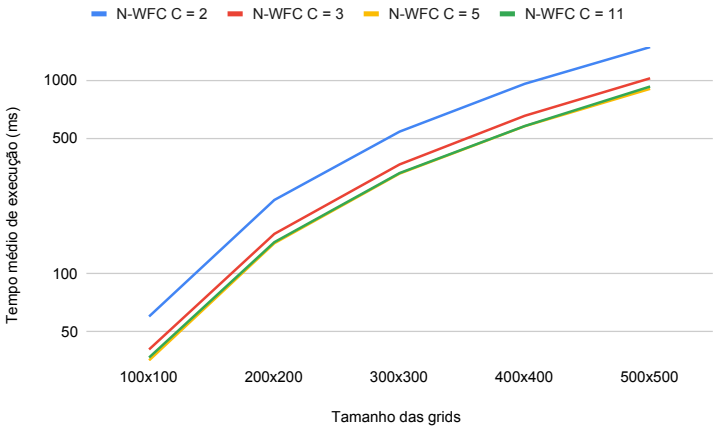

Figura 4. Tempo médio de execução (em ms) com base no tamanho da matriz, para o N-WFC variando o tamanho da subgrid em C++

Também foram realizadas medições sobre o uso de memória durante as execuções desses algoritmos. Como mostra o gráfico da Figura 5, o N-WFC com subgrids de tamanho 2 foi o algoritmo que mais consumiu memória, enquanto o DAC se mostrou o mais eficiente nesse quesito.

Para os experimentos conduzidos na *Unreal Engine*, os resultados são menos intuitivos. A Figura 6 apresenta o gráfico de tempo de processamento dos diversos algoritmos implementados. Diferentemente dos experimentos em C++, o N-WFC se mostrou mais rápido que o DAC e sua variante diagonal. Conjectura-se que esse comportamento se deve ao fato de que diversos outros métodos e funções competem pelos recursos computacionais disponíveis durante a execução da Unreal, prejudicando a memória CACHE.

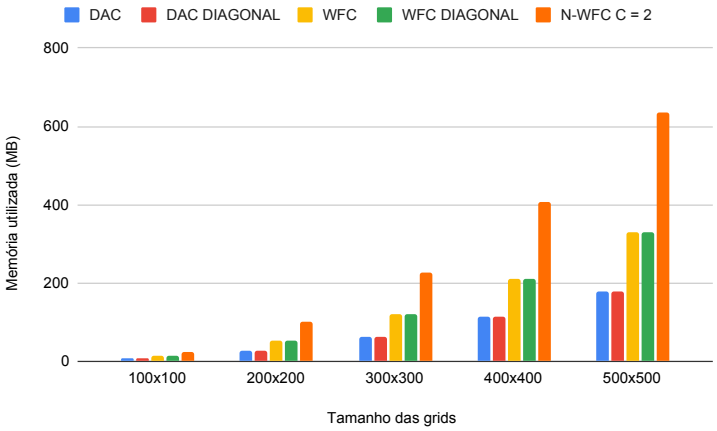

Figura 5. Uso de memória (em MB) com base no tamanho da matriz, para todos os algoritmos implementados em C++.

Por outro lado, o WFC original utilizando MRV, assim como na execução em C++ puro, se mostrou o método com menor desempenho. Esse resultado condiz com o esperado: o WFC original é um método genérico, que pode ser aplicado a *tilesets* incompletos e não tira proveito das especificidades do conjunto aqui usado. Por outro lado, no que se refere ao uso de memória, o algoritmo DAC é mais eficiente, conforme mostra o gráfico da Figura 7.

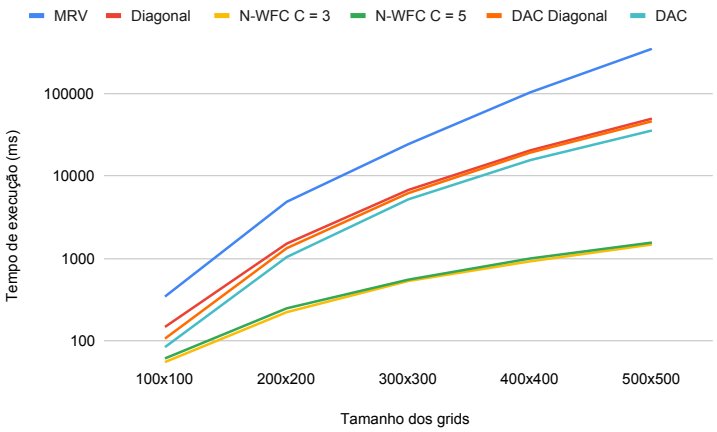

Figura 6. Tempo médio de execução (em ms) com base no tamanho da matriz, para todos os algoritmos implementados na *Unreal Engine*.

5. Conclusão

Neste artigo, o problema da geração procedural de terrenos bidimensionais e os algoritmos WFC e N-WFC foram analisados pela perspectiva da Programação por Restrições. O problema foi representado como uma instância CSP dual e sua estrutura mostrou o nível de consistência que garante solução sem *backtracking*. Dessa forma, compreendeu-se que o conceito de *tileset* sub-completo, introduzido por [Nie et al. 2023], visa atender a essa consistência. Com isso, foi proposta uma versão simplificada que faz a propagação de restrições de forma direcionada na matriz do terreno (DAC).

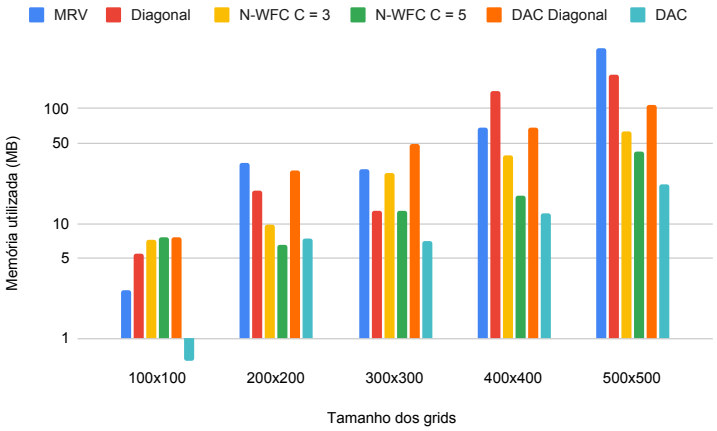

Figura 7. Uso de memória (em MB) com base no tamanho da matriz, para todos os algoritmos implementados na *Unreal Engine*.

Os resultados mostraram que o DAC, implementado diretamente em C++, melhora a eficiência do método. Porém, a escolha do algoritmo pode depender fortemente da plataforma, uma vez que na *Unreal Engine*, na qual diversos outros métodos e funções competem pelos recursos computacionais disponíveis, o DAC se saiu inferior ao N-WFC. Como trabalhos futuros, sugere-se uma análise de desempenho dos algoritmos executados na *Unreal Engine* para melhor compreender esse comportamento. Além disso, experimentos utilizando *tilesets* reduzidos, que não respeitam a consistência de caminho, podem ser conduzidos a fim de identificar o desempenho dos métodos quando ocorre o *backtracking*.

Por fim, considera-se que a principal característica da pesquisa em geração de mapas, em relação aos trabalhos clássico de CSP, é que no primeiro as aplicações (jogos) permite ao algoritmo especificar qual é o *tileset* necessário para que o algoritmo funcione de forma eficiente (i.e, sem *backtracking*), enquanto no caso geral de CSPs, o conjunto de domínios é dado pela instância.

Referências

Barrett, S. (2017). stb. Disponível em `https://github.com/nothings/stb`. Acessado em 28/06/2025.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32.

Gumin, M. (2016). Wave function collapse. Disponível em `https://github.com/mxgmn/WaveFunctionCollapse`. Acessado em 26/05/2025.

Kubi, G. (2021). Road tiles. Disponível em `https://kubigames.itch.io/road-tiles`. Acessado em 18/06/2025.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118.

Market.US (2024). Global generative ai in gaming market. Disponível em: https://market.us/report/generative-ai-in-gaming-market/. Acessado em 26/05/2025.

Mehta, N. (2025). The role of ai in game development and player experience. In *Proceedings of the International Conference on Innovative Computing Communication (ICICC 2024)*.

Merrell, P. C. (2021). Comparing model synthesis and wave function collapse. Disponível em https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf. Acessado em 18/06/2025.

Nie, Y., Zheng, S., Zhuang, Z., and Song, X. (2023). Extend wave function collapse algorithm to large-scale content generation. *arXiv preprint arXiv:2308.07307*.

Rose, T. J. and Bakaoukas, A. G. (2016). Algorithms and approaches for procedural terrain generation - a brief review of current techniques. In *8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, pages 1–2.

Rossi, F., van Beek, P., and Walsh, T., editors (2006). *Handbook of Constraint Programming*. Elsevier.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Shaker, N., Togelius, J., and Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer.

Togelius, J., Yannakakis, G. N., Stanley, K. O., and Browne, C. (2011). Search-based procedural content generation. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186.