UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Carmo de Oliveira

Estendendo WaveFunctionCollapse com grafos rotulados para a produção de conteúdo para jogos

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Carmo de Oliveira

Estendendo WaveFunctionCollapse com grafos rotulados para a produção de conteúdo para jogos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Márcia Aparecida Fernandes

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2024

Mateus Carmo de Oliveira

Estendendo WaveFunctionCollapse com grafos rotulados para a produção de conteúdo para jogos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Uberlândia, Brasil, 26 de abril de 2024:

Márcia Aparecida Fernandes
Orientador

Daniel Duarte Abdala

Rodrigo Sanches Miani

Uberlândia, Brasil
2024

Resumo

A geração procedural de conteúdo, GPC, é um recurso utilizado por muitos jogos digitais para fornecer aos jogadores um fluxo contínuo de experiências e interações novas. *WaveFunctionCollapse* é um algoritmo guloso sem *backtracking* de GPC, capaz de gerar mapas e texturas bidimensionais. Este trabalho tem como objetivo desenvolver uma extensão do algoritmo WFC, de tal forma que ela possa ser utilizada para produzir conteúdo em diferentes formatos, principalmente em formatos além da capacidade da implementação original. Para oferecer a capacidade e flexibilidade de descrição necessária para realizar isso, a estrutura central do algoritmo foi substituída por um multigrafo direcionado com arestas rotuladas. Uma implementação com as alterações propostas foi realizada e exemplos foram desenvolvidos para validar as funcionalidades, os quais mostraram que a extensão proposta é capaz de produzir uma maior gama de artefatos, com formatos mais variados e abrangentes que a implementação original, além de ainda reter a capacidade de produzir mapas e texturas bidimensionais.

Palavras-chave: Desenvolvimento de Jogos, Geração Procedural de Conteúdo, Algoritmos Heurísticos, Algoritmos Gulosos

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

Fig.	Figura
GPC	Geração Procedural de Conteúdo
WFC	*WaveFunctionCollapse*

Sumário

1 Introdução

A geração procedural de conteúdo, também chamada de GPC ou PCG (do inglês *Procedural Content Generation*), se refere à criação algorítmica, ao contrário de manual, de algum produto final que pode ser utilizado independentemente do sistema que o criou. A GPC pode ser utilizada para a definição de parâmetros de execução para um programa, a construção de ambientes virtuais, a criação de texturas, tabuleiros, música e diversos outros tipos de conteúdo.

Por meio da GPC, é possível obter uma maior flexibilidade do conteúdo que é utilizado para uma aplicação, principalmente devido à possibilidade de reduzir a quantidade de material que deve ser preparado de antemão para seu funcionamento. Por exemplo, a aplicação de um método que faz uso de algoritmos de preenchimento de espaço, que geram ambientes virtuais em tempo real, pode reduzir a complexidade de um projeto de simulação de um ambiente de realidade virtual e facilitar sua produção, como demonstrado no trabalho de Mattioli et al. (2015).

No cenário de jogos, GPC é utilizada para a criação de conteúdo em diversas escalas, desde materiais elementares para a criação de um jogo, como texturas, efeitos sonoros ou níveis, até sistemas complexos com os quais os jogadores interagem, como o comportamento de inimigos, ou as interações entre facções em um jogo de estratégia. Em termos práticos de desenvolvimento, GPC possibilita equipes menores de desenvolvedores criar conteúdo para seus projetos de forma mais rápida e em maior escala; Carli et al. (2011) sugere que a produção do material necessário para os projetos pode ser delegada aos métodos de GPC, com o auxílio de artistas e *designers* para controlar, apurar e consumar os resultados, o que pode acelerar o desenvolvimento.

O uso de GPC na perspectiva do *design* de jogos também levou ao surgimento de jogos que utilizam da natureza procedural dos artefatos gerados para oferecer experiências novas aos jogadores. Ainda antes da popularização dos computadores como produtos domésticos, GPC foi utilizada em muitos jogos de tabuleiro analógicos como *Dungeons & Dragons* para oferecer aos jogadores formas de produzir, ou auxiliar na produção de conteúdo novo por meio de algoritmos e procedimentos executáveis por humanos (SMITH, 2015). Jogos digitais podem utilizar mecânicas que envolvem GPC para controlar a dificuldade, ter um fluxo contínuo de conteúdo novo para ser explorado, ou realçar a experiência do jogador por meio de uma jogabilidade que se adapta a como ele joga. Dois dos primeiros jogos digitais a oferecer GPC como mecânica central foram *Elite* (1984), que gera e simula sistemas solares para serem navegados pelo jogador em sua nave, e *Rogue* (1980), que produz uma masmorra nova, populada com itens e inimigos aleatórios, em

cada jogatina.

Dentre os métodos de GPC para jogos, o algoritmo *WaveFunctionCollapse* (GUMIN, 2017) se tornou bastante popular. Baseado em métodos de síntese de texturas, WFC permite a geração de *bitmaps*, mapas bidimensionais, e outros tipos de saídas no formato de uma grade quadrada ou retangular, a partir de um conjunto discreto de "peças" que são dispostas de acordo com regras de adjacência. O algoritmo já foi utilizado em diversos jogos, como *Caves of Qud* (2015) e *Bad North* (2018), e estendido para ser capaz de produzir outros tipos de artefatos, como espaços tridimensionais e poesia.

A pesquisa na área de GPC é recente, e envolve múltiplas subáreas da Ciência da Computação, como também do *Design* e da Psicologia. Além de pesquisas sobre novos métodos de GPC e a otimização dos mesmos, há uma sub-área focada em pesquisar por maneiras de manipular os métodos tal que as pessoas que os utilizam, como *designers*, tenham mais controle sobre seu funcionamento, onde "controle" se refere a quão bem os parâmetros ou funcionamento geral pode ser manipulado para alcançar um fim desejado. No contexto do algoritmo WFC, isso se manifesta em desenvolvimentos sobre como facilitar a interação humana com o algoritmo, utilizando sistemas de iniciativa mista (LANGENDAM; BIDARRA, 2022), ou por meio de modificações do algoritmo para permitir que o usuário defina regras de geração mais elaboradas ou com mais funcionalidades (SANDHU; CHEN; MCCOY, 2019).

1.1 Motivação e Objetivos

O algoritmo WFC, apesar de popular e versátil para seu caso de uso padrão, é limitado à geração de produtos que podem ser representados em uma grade retangular ou quadrada. Isso dificulta a implementação do algoritmo na geração de conteúdo com outros formatos, e leva à necessidade de modificações e extensões específicas para mapear o resultado ou flexibilizar o algoritmo o suficiente para cada caso de uso.

O objetivo deste trabalho é desenvolver uma extensão para o algoritmo WFC que pode ser utilizada em cenários além daqueles abordados pela forma tradicional. A extensão que será proposta deverá ser capaz de produzir, além do conteúdo do qual o WFC tradicional é capaz, conteúdo de outros tipos e formatos, além de oferecer formas padronizadas de representar a estrutura desse conteúdo e regras de geração para eles. Para isso, a estrutura central do algoritmo será substituída por um multígrafo rotulado, e o algoritmo será modificado para que possa ser executado sobre esta estrutura.

Para alcançar este objetivo, este trabalho pretende:

- Analisar o algoritmo *WaveFunctionCollapse*, seu funcionamento e suas limitações.

- Desenvolver modificações ao algoritmo para subverter essas limitações, ao mesmo tempo que o funcionamento central do algoritmo é mantido.
- Desenvolver uma implementação destas modificações para que as mesmas possam ser validadas quanto ao seu funcionamento.

A implementação de teste da extensão, e os exemplos utilizados na validação, serão feitos nas linguagens de programação C e C++, e utilizarão a biblioteca gráfica *Raylib*[1], disponível no Repositório GitHub por Ramon Santamaria.

1.2 Organização da Monografia

Este trabalho será dividido em 5 capítulos: Introdução, Revisão Bibliográfica e Fundamentação, Desenvolvimento, Resultados e Conclusão.

O Capítulo 2 descreverá a fundamentação conceitual do trabalho. Os principais conceitos teóricos serão abordados, além de uma análise e descrição do algoritmo *Wave-FunctionCollapse*. Em seguida, alguns trabalhos relacionados serão analisados, com foco sobre como eles modificaram, estenderam ou aplicaram o algoritmo WFC.

O Capítulo 3 detalhará como o trabalho terá sido desenvolvido. O *design* principal da extensão será descrito, com observações sobre as modificações no algoritmo e nas estruturas de dados envolvidas, e um comparativo entre o algoritmo original e a extensão será realizado.

No Capítulo 4, a extensão proposta será validada. Os exemplos usados na validação serão descritos em termos de como fazem uso das funcionalidades fornecidas pela extensão, e os resultados gerados serão mostrados. Após isso, uma discussão sobre as capacidades da extensão e as consequências de seu uso nos exemplos será realizada.

Por fim, o Capítulo 5 oferecerá considerações finais e conclusões sobre a extensão e o trabalho. Aspectos do *design* e dos testes que poderiam ser explorados em trabalhos futuros também serão levantados.

[1] https://github.com/raysan5/raylib

2 Fundamentação e Revisão Bibliográfica

Neste capítulo, a fundamentação teórica é apresentada, com tópicos relacionados ao algoritmo *WaveFunctionCollapse*, métodos de geração procedural de conteúdo e grafos, utilizados ao decorrer do trabalho. Em seguida, uma revisão de trabalhos que fazem uso do WFC é apresentada.

2.1 Tecnologia Utilizada

Nesta seção, primeiramente, definições e conceitos importantes em relação a grafos são introduzidas. Em seguida, uma taxonomia dos principais métodos de GPC é descrita. Por fim, o algoritmo principal utilizado no trabalho, *WaveFunctionCollapse*, é descrito em relação ao seu funcionamento e à taxonomia apresentada.

2.1.1 Grafos

Um *grafo* é uma construção matemática que consiste em uma conjunto de objetos, denominados "nós", e um conjunto de pares de nós. Comumente, os pares de nós em um grafo descrevem alguma relação entre eles no contexto em que é usado, que associa um nó ao outro e vice-versa, e são referidos como "arestas". Uma representação visual de um grafo pode ser visto na Figura 1.

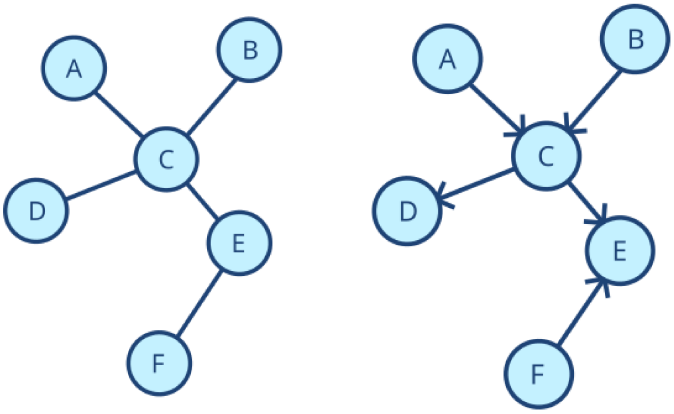


Figura 1 – Um par de grafos, um direcionado, e um não direcionado, respectivamente.

Matematicamente, um grafo G pode ser descrito como $G = (V, E)$, onde V consiste em um conjunto de nós e $E \subseteq \{(x, y) \mid (x, y) \in V^2,\ x \neq y\}$, isto é, E é um subconjunto do conjunto de todos os pares de nós menos aqueles onde ambos os elementos no par são o mesmo nó. Caso os pares em E sejam ordenados (isto é, $(u, v) \neq (v, u)$ para $u \neq v$), o grafo é dito orientado ou direcionado, e implica que uma aresta que vai de um nó u para um v não pode ser seguida no caminho oposto; ou seja, não há necessariamente

um caminho de v para u no caso. Também na Figura 1 é demonstrado visualmente esta distinção.

O grau de um nó u em um grafo G, dado como $d_G(u)$ ou $d(u)$, descreve a quantidade de arestas incidentes em u. Em um grafo comum, este número é dado pelo número de arestas nas quais uma das pontas se dá no nó u, enquanto em um grafo direcionado apenas as arestas que saem de um nó e chegam em u são contadas (isto é, os pares (x, u), onde x é algum outro nó do grafo). Dois nós em um grafo não-direcionado são ditos adjacentes se há uma aresta entre eles, enquanto em um grafo direcionado, um é adjacente do outro se há uma aresta que parte do primeiro e chega no segundo (no caso, caso não haja uma aresta no caminho contrário, o segundo não é adjacente do primeiro).

Um grafo normal não permite a presença de mais de uma aresta entre dois nós, ou, no caso de grafos direcionados, mais de uma aresta que parte e chega dos mesmos nós. Grafos que permitem a ocorrência de mais de uma aresta são denominados multígrafos. A Figura 2 distingue visualmente um grafo de um multígrafo. Nota-se também que tanto grafos quanto multígrafos não permitem arestas que partem e chegam no mesmo nó.

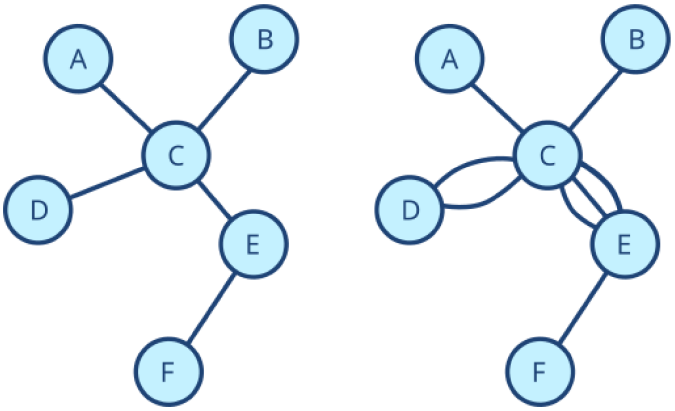


Figura 2 – Um grafo e um multígrafo, respectivamente.

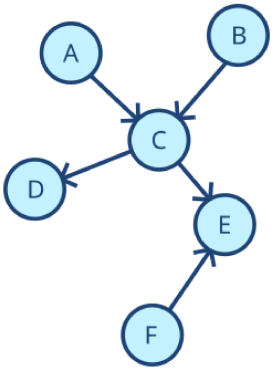
Grafos podem ser representados em código por meio de estruturas de dados especializadas. Há três principais métodos para realizar tal representação: uma matriz de adjacência, uma matriz de incidência e listas de adjacência. Um visual destas estruturas se encontra na Figura 3.

Uma matriz de adjacência envolve a criação de uma matriz (ou um vetor de vetores) na qual cada linha corresponde a um nó, e cada coluna a outro. Cada entrada da matriz em uma linha u e coluna v representa um valor dado à uma aresta entre os nós u e v. Comumente, um valor de zero na posição da matriz representa que não há uma aresta, enquanto qualquer outro valor indica a existência de uma aresta entre os nós. Por conta da construção da matriz, um grafo não direcionado tem uma matriz M simétrica, uma vez que para nós u e v, $M[u, v] = M[v, u]$.

Uma matriz de incidência igualmente envolve a criação de uma matriz. Este método

se difere da matriz de adjacência ao representar nas colunas da matriz as arestas do grafo. Um valor em uma linha u e coluna e deste tipo de matriz representa se o nó u faz parte da aresta e, ou seja, indica se e parte ou chega em u.

Uma lista de adjacência consiste em criar uma estrutura ou objeto para cada nó do grafo, e, dentro deste, armazenar uma lista de nós adjacentes. Desta forma, cada nó armazena com quais outros nós ele forma uma aresta, além de ser possível armazenar informações adicionais sobre o nó na mesma estrutura. Este método é principalmente útil em grafos de tamanho menor; em grafos grandes, com uma quantidade substancial de nós e arestas, uma matriz de adjacência pode trazer benefícios em termos de uso de espaço de memória e tempo de acesso.



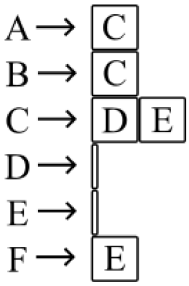
(a) Grafo de exemplo

	A	B	C	D	E	F
A	0	0	1	0	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	1	0

(b) Matriz de Adjacência

	$\overrightarrow{AC}$	$\overrightarrow{BC}$	$\overrightarrow{CD}$	$\overrightarrow{CE}$	$\overrightarrow{FE}$
A	1	0	0	0	0
B	0	1	0	0	0
C	1	1	1	1	0
D	0	0	1	0	0
E	0	0	0	1	1
F	0	0	0	0	1

(c) Matriz de Incidência



(d) Lista de Adjacência

Figura 3 – Representação de um grafo como estruturas de dados.

2.1.2 Métodos de GPC

Na geração procedural de conteúdo, há diversos métodos que podem ser utilizados para construir um programa que produz um produto desejado, também chamado de "artefato", e cada um faz uso de tecnologias e conceitos distintos em sua execução. Compton (2017) elabora uma forma de classificar métodos de GPC de acordo com como eles desenvolvem o artefato desde seu estado inicial até um estado utilizável. Ela separa-os em duas principais categorias: métodos aditivos e subtrativos.

Métodos aditivos são algoritmos onde um artefato é construído incrementalmente. Uma das formas mais simples destes métodos são os métodos de peças, onde peças determinadas previamente são agrupadas, de acordo com regras e restrições, para formar um produto. Outros exemplos de métodos nesta categoria são: gramáticas formais, que podem ser usadas para construir estruturas a partir de um símbolo inicial por meio de regras de construção; métodos de parametrização, nos quais um objeto base é modificado por meio de parâmetros randomizados para produzir uma variação nova; e métodos de simulação, onde um conjunto de dados é desenvolvido com algum algoritmo ao decorrer

de um número de iterações até que obtenha-se um resultado satisfatório de acordo com uma métrica ou teste.

Métodos subtrativos, por outro lado, envolvem a geração de vários artefatos, seguida do uso de algum filtro para remover artefatos inválidos ou que possuem alguma característica indesejada. A forma mais simples de um método subtrativo seria uma algoritmo que gera todo o espaço amostral de um artefato (isto é, todas as variações possíveis ou parte delas) e depois as filtra por meio de alguma regra. Métodos de busca, como os utilizados na área de Inteligência Artificial, e métodos de satisfação de restrições são métodos subtrativos, uma vez que envolvem buscar um artefato que atenda aos requisitos e ignorar ou eliminar os outros.

Nota-se que, dentro da GPC, o uso de não-determinismo nos métodos não é necessário. Pelo contrário, métodos de GPC podem ser completamente determinísticos. A utilidade do determinismo se dá em cenários onde uma descrição de algum artefato é menor ou menos trabalhosa que o artefato em si. Em sua *survey*, Togelius et al. (2011) comenta sobre o jogo *.kkrieger*, que utiliza GPC para produzir suas texturas e modelos tridimensionais no momento de execução; graças a isso, o jogo é particularmente pequeno, com um arquivo executável de apenas 95KB. Dessa forma, "algoritmos de GPC completamente determinísticos podem ser vistos como uma forma de compressão de dados" (TOGELIUS et al., 2011).

Pode-se também classificar os métodos em relação ao tipo de artefato que é produzido. Em sua *survey*, Hendrikx et al. (2013) propõem distinguir os tipos de conteúdo produzidos em algoritmos de GPC para jogos em 5 categorias principais: pedaços (*game bits*), espaços, sistemas, cenários, *design*, e conteúdo derivado.

Texturas, efeitos sonoros e visuais, e outras partes de detalhe que compõem elementos maiores são pedaços, enquanto espaços detalham quais pedaços são usados e como eles são dispostos, como mapas. Sistemas são ecossistemas, redes urbanas e comportamentos de personagens não-jogáveis no mundo do jogo. Cenários envolvem componentes que controlam a ordem de eventos em um jogo, como a história, eventos, sequências de níveis e quebra-cabeças. *Design* constitui as regras de um jogo, e como o jogador interage com os mesmos. Por fim, conteúdo derivado se refere à conteúdo que é gerado a partir do que ocorre em um jogo, sem necessariamente ser parte do jogo, como anúncios de comunidade em redes sociais sobre eventos dentro do jogo, ou placares de pontuação. A Figura 4 relaciona estas categorias como uma pirâmide: cada tipo de artefato é mais complexo e abstrato, e, muitas vezes, uns são utilizados como base para aqueles no nível superior.

Na separação de Hendrikx, um método pode se encontrar em diversas categorias. Técnicas que se baseiam em geração de números pseudo-aleatória (PRNG), por exemplo, podem ser utilizadas para gerar texturas (por meio de ruído de Perlin), mas também podem ser usadas para escolher trajetos aleatórios de uma história, ou servirem como

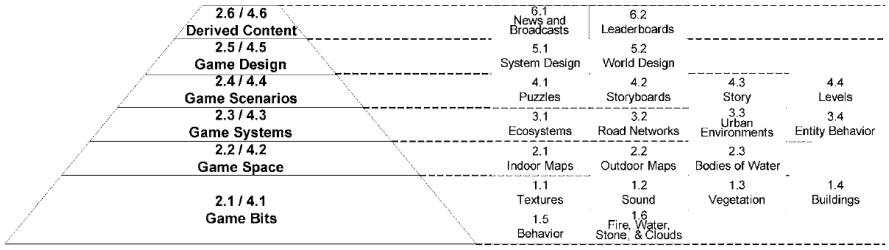


Figura 4 – As cinco categorias de artefato descritas. Fonte: Extraído de (HENDRIKX et al., 2013)

estrutura para uma mecânica ou sistema que encompasse o jogo inteiro. Sistemas de Lindenmayer, um tipo de gramática formal, podem ser usados para descrever a construção de uma árvore (Figura 5), o que seria classificado como um "pedaço", como também uma rede de ruas e avenidas de uma cidade, o que seria um "sistema".

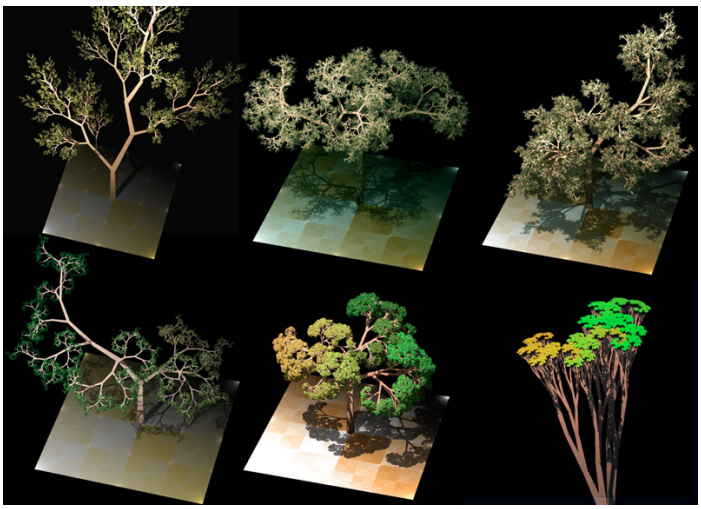


Figura 5 – Um conjunto de árvores geradas por Sistemas de Lindenmayer. Cada ramificação, folha e extensão dos ramos é definida por uma regra na gramática formal utilizada, e a árvore é construída a partir de um símbolo inicial.
Fonte: Wikipedia[1].

2.1.3 *WaveFunctionCollapse*

O algoritmo *WaveFunctionCollapse* é um algoritmo de GPC que, na classificação proposta por Compton (2017), se enquadra como um método subtrativo. O principal objetivo do algoritmo é a geração de texturas, mapas bidimensionais ou, em seu nível mais básico, artefatos no formato de grade retangular ou quadrada. A Figura 6 mostra um exemplo de um artefato que pode ser gerado, e o que é utilizado em sua geração.

Em sua forma tradicional, o algoritmo recebe um conjunto de elementos, chamados de ladrilhos (do inglês *tiles*) ou peças, uma série de regras, que especificam onde quais

[1] https://en.wikipedia.org/wiki/L-system

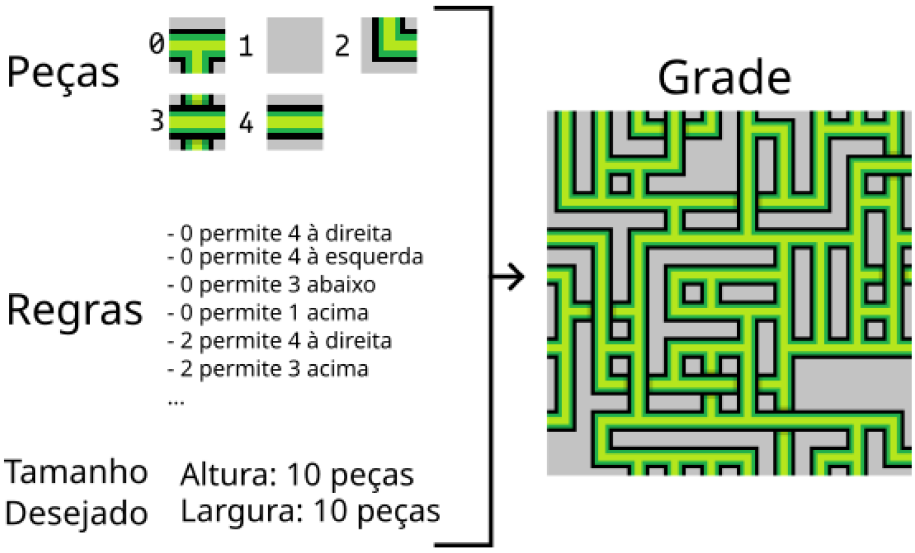


Figura 6 – Entradas e saída do algoritmo WFC. Neste caso, o resultado gera peças na grade com os tubos verdes devidamente conectados.

peças podem ser colocadas em relação a outras, e o tamanho da saída desejada em altura e largura. O algoritmo inicializa conjuntos de possibilidades (no caso, as peças possíveis) para cada espaço do artefato que será produzido e, de acordo com as regras, realiza rodadas de restrição das possibilidades nestes conjuntos até que as possibilidades sejam reduzidas até uma escolha única. O resultado da saída do algoritmo é um espaço do tamanho requisitado preenchido com peças de tal forma que as regras e relações dadas sejam seguidas. Ele se baseia em um ciclo de duas operações básicas, *Observação* e *Propagação*, como pode ser visto no Algoritmo 1.

Algoritmo 1: O ciclo de execução principal do *WaveFuncionCollapse*

Entrada: altura, largura, peças, regras
Saída : grade preenchida
onda ← **GerarCélulas**(altura, largura, peças)
propagador ← **GerarPropagador**(peças, regras)
pilha de propagação ← pilha vazia
enquanto *houverem células ainda não colapsadas*
  Observar(onda, pilha de propagação)
  PropagarMudanças(onda, propagador, pilha de propagação)
fim
return ExtrairGrade(onda)

Inicialmente, o algoritmo gera um conjunto denominado "onda" (*wave*), composto por "células". Cada célula representa um espaço no qual uma peça poderá se situar na saída final, e mantém uma lista de peças que podem, na iteração atual do algoritmo, ser colocadas nela. A onda é gerada no início da execução com (altura × largura) células e, como nenhuma restrição ainda é aplicada no primeiro momento, elas são inicializadas com listas preenchidas com todas as peças, como pode ser visto na Figura 7.

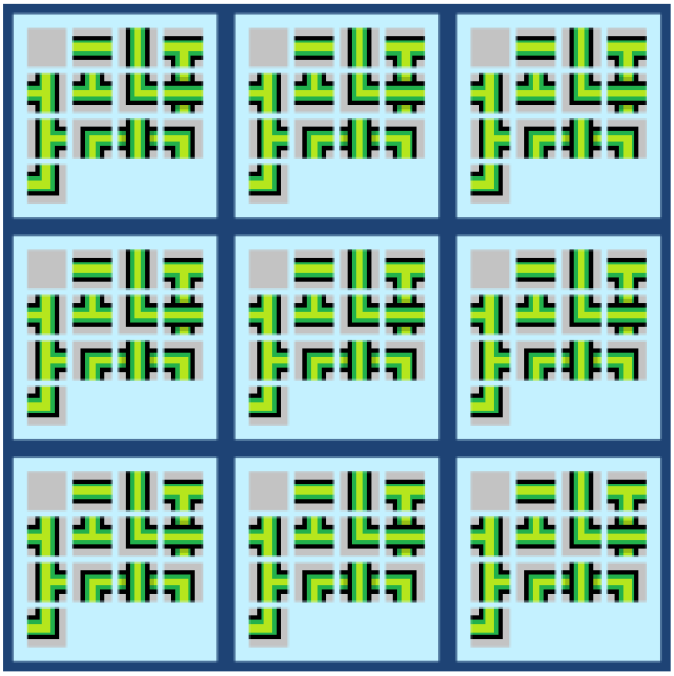


Figura 7 – Uma onda de tamanho 3×3 em seu estado inicial, com o conjunto de peças da Figura 6. As células são representadas em azul, e as peças dentro delas representam as peças disponíveis em cada.

Além da onda, as regras são lidas e codificadas. Tipicamente, as regras são descritas no algoritmo como triplas (p_1, p_2, dir), para representar que a peça p_1 permite a peça p_2 na direção dir, onde uma "direção" se refere à uma das quatro direções cardinais na grade: cima, baixo, esquerda ou direita. As regras também podem ser geradas a partir de um exemplo: uma grade de entrada já preenchida é dada, e as peças presentes são utilizadas para descrever quais peças podem ser colocadas ao lado de quais e em quais direções. As regras são compiladas em uma estrutura chamada "propagador", que facilita o resgate das regras durante a execução do algoritmo. Comumente, o propagador é uma tabela *hash* ou outra estrutura de dados de rápida indexação que mapeia cada peça às peças que ela permite em cada direção, mas pode também ser simplesmente um vetor com as regras que é percorrido linearmente.

Por fim, inicializa-se também uma pilha de propagação. Esta estrutura de dados é utilizada para armazenar quais células foram modificadas, para que suas mudanças sejam propagadas para as células adjacentes na onda.

Após a inicialização, o algoritmo realiza uma observação. Nesta operação, uma célula é escolhida e uma peça de sua lista de peças disponíveis é escolhida aleatoriamente. Opcionalmente, cada peça pode possuir um peso, e a escolha aleatória é ponderada. A peça escolhida passa a ser a peça final para a célula em questão, e todas as outras deixam de ser opções válidas; a célula é dita "colapsada" ou "observada". A escolha de qual célula sofre colapso em cada observação depende de uma heurística de entropia mínima: a célula com menor entropia em relação à probabilidade de ocorrência das peças disponíveis em

sua lista será escolhida. A probabilidade de ocorrência de cada peça é relativa ao seu peso; quando as peças não possuem pesos associados, a probabilidade é igual para todas as peças. Matematicamente, a célula com menor H é escolhida, onde H é descrito pela equação 2.1.

$$H = \sum_{t \in T_C} P(t) \cdot \log P(t) \tag{2.1}$$

Nessa equação, $P(t)$ representa o peso da peça t, e T_C é o conjunto de peças possíveis para a célula C. Como células já colapsadas não possuem opções (entropia nula), elas podem ser ignoradas na escolha. O Algoritmo 2 descreve esse procedimento.

Algoritmo 2: Lógica da Observação

Entrada: onda, pilha de propagação
Saída : onda com uma peça colapsada, pilha de propagação modificada

```
1  min ← ∞
2  argmin ← −1
3  para cada célula na onda faça
4  |  se célula colapsada então
5  |  |  continue
6  |  fim
7  |  entropia ← 0
8  |  para cada peça p na célula faça
9  |  |  entropia ← entropia + P(p) · log P(p)
10 |  fim
11 |  se entropia < min então
12 |  |  min ← entropia
13 |  |  argmin ← célula
14 |  fim
15 fim
16 se argmin < 0 então
17 |  marcar onda como finalizada
18 |  retornar
19 fim
20 senão
21 |  collapsedVal ← uma peça na lista da célula em argmin, escolhida
   |    aleatoriamente
22 |  remover todas as peças na lista de argmin, exceto collapsedVal
23 |  marcar argmin como colapsada
24 |  adicionar argmin no topo da pilha de propagação
25 fim
26 retornar onda, pilha de propagação
```

Note nas linhas 16 e 17 do pseudocódigo acima que um `argmin` negativo implica que nenhuma célula foi selecionada para colapso. Isso indica que todas as células possuem

um valor definitivo, e o *loop* principal do algoritmo pode ser encerrado. Por isso, a onda é marcada como finalizada.

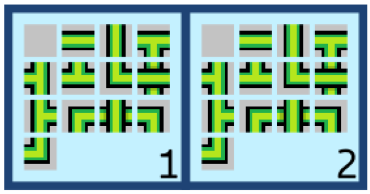
Com a observação, uma célula entra em um estado definitivo, o que implica que as células ao redor da colapsada podem possuir peças em suas listas que não são, pelas regras definidas, permitidas em suas posições. Para que o estado das células da onda não contradiga as restrições, a alteração deve ser propagada para os seus arredores: a célula é adicionada na pilha de propagação para ser tratada.

Para realizar a propagação de mudanças de uma célula de origem para uma de destino, verificam-se quais peças estão na lista no destino, e a direção da origem ao destino. Caso não haja uma peça na lista da célula de origem que permite uma peça na lista da de destino, esta peça no destino é banida, ou seja, é removida da lista. Esse processo é repetido com todas as peças na lista da célula de destino. Se a lista na célula de destino for alterada, esta alteração também deve ser propagada, e por isso ela é adicionada na pilha de propagação. Um pseudocódigo de execução é descrito no Algoritmo 3.

Algoritmo 3: Lógica da Propagação.

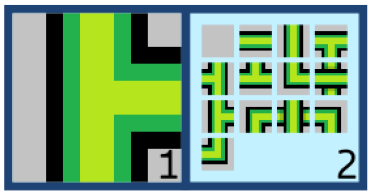
```
   Entrada: onda, pilha de propagação, propagador
   Saída  : onda com as mudanças propagadas, pilha de propagação vazia
 1 enquanto houverem células na pilha de propagação
 2     célulaAtualizada ← remover célula no topo da pilha de propagação (pop)
 3     para cada direção dir faça
 4         célulaAdjacente ← célula adjacente a célulaAtualizada na direção dir.
 5         se célulaAdjacente já colapsada então
 6             continue
 7         fim
 8         para cada peça pAdj na lista de célulaAdjacente faça
 9             permitida ← false
10             para cada peça pAtu na lista de célulaAtualizada faça
11                 se pAtu permite pAdj na direção dir então
12                     permitida ← true
13                     break
14                 fim
15             fim
16             se permitida = false então
17                 remover pAdj da lista de célulaAdjacente
18             fim
19         fim
20         se quantidade de peças na lista de célulaAdjacente mudou então
21             adicionar célulaAdjacente na pilha de propagação
22             marcar célulaAdjacente como colapsada se quantidade de peças na
                lista = 1
23         fim
24     fim
25 fim
26 retornar onda, pilha de propagação
```

A Figura 8 mostra a propagação entre um par de células. Para este conjunto de peças, as regras apenas permitem peças adjacentes se elas se conectarem visualmente (Figura 6). A Figura 9 mostra a propagação inteira em uma grade 5×5.



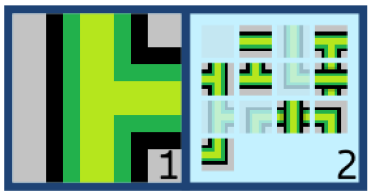
[]

(a) As duas células iniciam com todas as opções de peças em suas listas.



[1]

(b) A célula 1 é observada, e é adicionada na pilha de propagação.



[2]

(c) As mudanças são propagadas, e algumas peças são banidas da lista da célula 2. Agora, célula 2 é adicionada na pilha.

Figura 8 – Um exemplo de propagação entre duas células. Células banidas são representadas como transparentes.

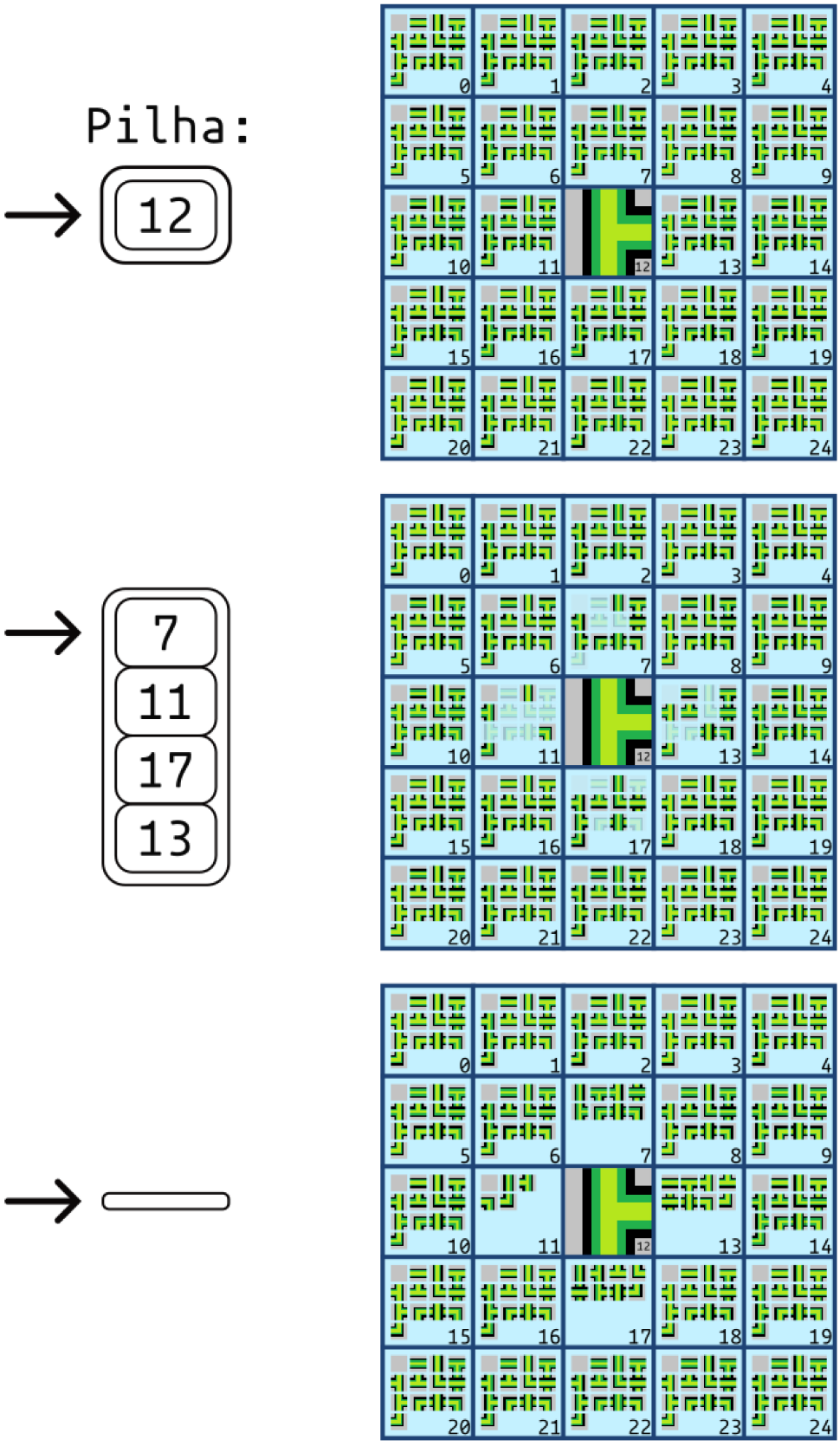


Figura 9 – Demonstração de uma propagação. Assume-se que a ordem das direções é direita, baixo, esquerda e cima. Note que a propagação das células 7, 11, 17 e 13 não modifica as células ao redor, logo nenhuma nova célula é adicionada na pilha.

O algoritmo realiza observações e propagações até que todas as células da onda tenham sido colapsadas. Porém, existem ocasiões onde o algoritmo, por sua natureza estocástica, realiza uma série de colapsos que resultam em células sem nenhuma peça válida possível, como pode ser visto na Figura 10: na onda à esquerda, temos tubos no formato de "L" no conjunto de peças, e a observação das duas células adjacentes restringe as opções para um desses tubos; porém, na onda à direita, como não temos peças em "L", se as duas peças adjacentes colapsarem como na Figura, a célula em cinza escuro ficará sem nenhuma peça válida em sua lista.

Estes casos são denominados "contradições" e, podem ser tratados com uma etapa de *backtracking*, na qual a última observação é revertida e outra peça é escolhida para o colapso. Caso o *backtracking* seja complexo ou computacionalmente caro, pode-se optar por um reinício global, ou seja, o descarte do estado atual do algoritmo e a reinicialização da onda, uma vez que o algoritmo, "em prática, tende a encontrar contradições surpreendentemente raramente" (GUMIN, 2017).

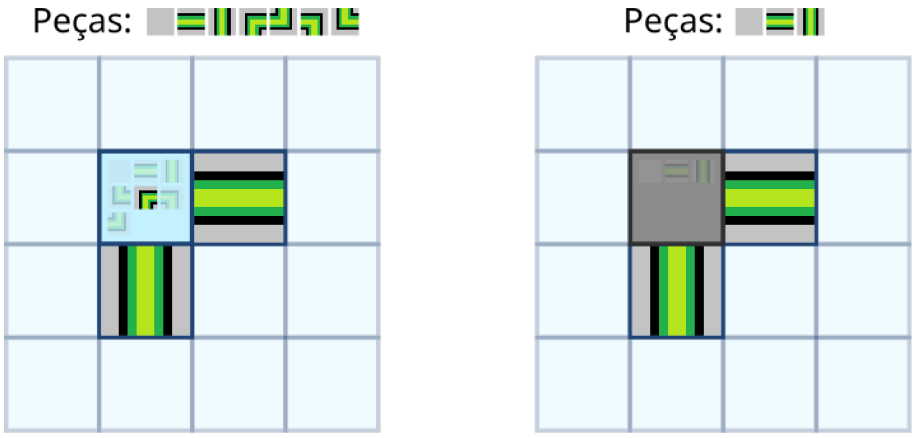


Figura 10 – Exemplo de contradição. Note na onda à direita como nenhuma das peças pode ser colocada na célula em cinza escuro.

O resultado da execução bem-sucedida do algoritmo é uma grade preenchida com uma única peça em cada espaço, e pode ser extraída diretamente da onda (que, no fim, será composta inteiramente por células colapsadas contendo apenas uma única peça). O que as peças representam no artefato final depende do contexto da aplicação: em sua implementação original, o algoritmo opera com o intuito de produzir uma textura, onde cada peça representa um *pixel* final, ou um mapa de jogo retangular, onde cada peça representa um pedaço do mapa (uma parede, uma escada, um pedaço de chão, etc.). Porém, é possível representar outros tipos de conteúdo. *Oísin*[2] é uma implementação do algoritmo disponível no repositório GitHub com o intuito de gerar poesia, onde as peças são palavras e a grade são os versos de métrica fixa do poema.

Há uma outra forma do algoritmo que trabalha com padrões de peças ao invés de

[2] https://github.com/mewo2/oisin

peças individuais. Denominado WFC sobreposto (em inglês, *Overlapping WFC*), esta variação do algoritmo exige obrigatoriamente um exemplo de entrada. Ao invés de trabalhar diretamente sobre as peças, grupos de peças de tamanho $N \times N$ no exemplo são extraídos, e cada padrão único encontrado é tratado como uma peça separada. As regras são geradas de acordo com quais padrões se sobrepõem em cada direção. Fora este pré-processamento e o uso de peças que representam grupos únicos de peças, o algoritmo utiliza as mesmas operações que o WFC simples. Segundo Gumin (2017), essa forma do algoritmo é a forma principal do mesmo, e o WFC que trabalha com peças individuais é o caso onde $N = 1$. A Figura 11 mostra o processo de geração nessa variação.

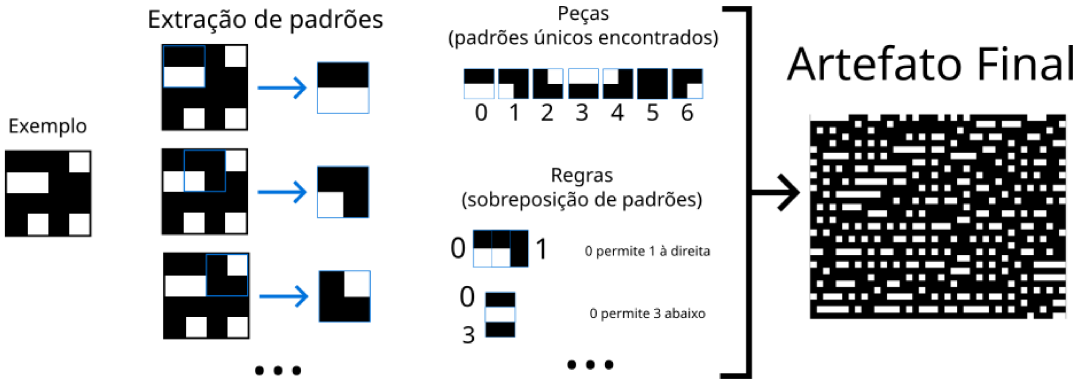


Figura 11 – A leitura dos padrões de um exemplo, e como eles geram as peças e regras utilizadas no WFC.

A versatilidade e controle sobre o funcionamento do algoritmo foi responsável por sua popularização entre desenvolvedores de jogos. Karth e Smith (2017) comenta sobre como WFC é comumente usado como uma "caixa preta", no sentido de que entender o funcionamento interno do algoritmo não é necessário para utilizá-lo, e também descreve o algoritmo como "orientado à dados", pela habilidade de controlar os artefatos gerados apenas mudando os dados de entrada. Como é possível gerar as regras para o algoritmo a partir de um exemplo usado como entrada (sobre o qual as regras entre peças são extrapoladas), e graças à performance razoável, pode-se ver como ele alcançou sua atual popularidade.

2.2 Trabalhos Relacionados

Nesta seção, alguns trabalhos realizados sobre WFC são apresentados, que oferecem outras perspectivas e ideias sobre o algoritmo que foram úteis para a realização do trabalho.

2.2.1 WFC como Satisfação de Restrições

Karth e Smith (2017), em seu artigo sobre o algoritmo WFC, propõe que o mesmo possui muitas similaridades com problemas de satisfação de restrições. Eles demonstram

uma reformulação do algoritmo por meio de Programação de Conjunto de Respostas (ASP, do inglês *Answer Set Programming*), um paradigma de programação declarativa que estabelece um modelo para um problema com um caso de falha bem definido, que pode então ser usado para buscar soluções válidas para o problema (isto é, soluções que são verdadeiras no modelo). Eles utilizam *Clingo*, um programa de ASP focado em problemas combinatoriais.

Apesar de não modelarem a heurística de entropia no programa, eles mostram em seus resultados que mesmo sem o uso de uma heurística o algoritmo é capaz de resolver e encontrar uma solução (isto é, produzir um artefato final) com poucas falhas. Porém, eles também mostram que, ao utilizar a ideia do algoritmo original de simplesmente reiniciar toda a geração no momento que uma falha ocorre, o algoritmo falha mais vezes em sua execução quando restrições globais (isto é, restrições gerais aplicadas a toda a onda, não limitadas a restrições entre peças), o que reduziu severamente a eficiência do algoritmo. Nos testes realizados por eles, o algoritmo foi dado uma janela de um minuto para produzir um artefato, e, após a adição das restrições globais, ele não conseguiu mais produzir o artefato à tempo.

Karth e Smith (2017) argumentam que, em cenários reais, restrições globais podem ser vitais para garantir que *designers* tenham controle sobre o que é produzido pelo algoritmo. Para isso, segundo eles, é necessário encontrar um equilíbrio entre o uso de reinícios totais e *backtracking* para que a eficiência do algoritmo possa ser mantida bem em aplicações do algoritmo.

2.2.2 Métodos de Iniciativa Mista

Em seu artigo, Langendam e Bidarra (2022) fazem uso de conceitos de inteligência artificial e interação humano-computador para propor uma ferramenta chamada *miWFC*. A ferramenta desenvolvida oferece uma série de ferramentas para auxiliar no uso direto do algoritmo WFC em *design*.

A ferramenta deriva seu nome da palavra *mixed-initiative*, ou iniciativa mista. Métodos de iniciativa mista são um ramo da área de inteligência artificial que busca aperfeiçoar a interação de usuários humanos com sistemas de agentes inteligentes. No caso de *miWFC*, os autores propõem uma interface gráfica guiada (GUI, do inglês *Guided User Interface*) pela qual um *designer* humano poderia interagir com o funcionamento do algoritmo WFC e controlar o seu funcionamento com maior clareza.

O programa desenvolvido oferece a possibilidade de voltar e avançar passo-a-passo no processo do algoritmo, além de oferecer ferramentas para que o usuário possa diretamente manipular partes artefato em produção, como uma ferramenta "lápis" que permite o colapso manual de células, ou um "pincél" que reverte o colapso de uma região de células.

Com o uso destas um *designer* conseguiria manipular o resultado do algoritmo durante a execução do mesmo com boa granularidade.

2.2.3 Restrições de *design* sobre WFC

A forma tradicional de *WaveFunctionCollapse* trabalha apenas com restrições relativas à adjacência de células na onda: as restrições sobre quais peças podem ocorrer em quais locais são baseadas inteiramente na direção imediata entre pares de células. Porém, fora este tipo de restrição local, outras restrições que podem ser úteis para o controle da geração não podem ser representadas diretamente no WFC. Sandhu, Chen e McCoy (2019) abordam em seu trabalho a falta de uma integração do que eles chamam de "restrições de *design*" no algoritmo, e propõem modificações, focadas no âmbito de geração de mapas para jogos, para que seja possível implementá-las.

Os autores descreveram três restrições de *design*: restrições não-locais, onde o colapso de uma célula causa o colapso de uma região de células na onda, ao invés de apenas as quatro direções imediatamente adjacentes; recálculo de peso, onde o peso de peças em uma região da onda pode mudar dinamicamente; e a propagação em área, onde uma célula pode propagar restrições para uma área dinâmica de células ao invés de apenas as células adjacentes.

Para implementar estas restrições, o ciclo principal do algoritmo WFC é estendido para incluir etapas adicionais de observação ou propagação que são ativadas de acordo com as células colapsadas. Quando uma célula colapsar em uma peça que deve causar um recálculo de pesos, por exemplo, a etapa no ciclo é percorrida de acordo com parâmetros especificados no início da execução.

Os autores também realizaram um comparativo entre cada restrição e o WFC tradicional, em termos de tempo de execução e uso de memória. Segundo Sandhu, Chen e McCoy (2019), as restrições levaram a um aumento em ambos os aspectos verificados, com o recálculo de pesos das peças responsável pelo maior aumento tanto no tempo de execução quanto da memória utilizada. Também foi identificado que a propagação em área aumentou muito o número de contradições que ocorriam na execução do algoritmo, o que dificultou a coleta de dados sobre essa restrição.

O artigo oferece uma forma nova de implementar restrições comuns no *design* de algoritmos de GPC no *WaveFunctionCollapse*. Apesar dos custos adicionais, as restrições oferecem um grau de controle sobre o artefato gerado que o algoritmo tradicional não é capaz de oferecer, o que é útil para aperfeiçoar o conteúdo antes de utilizá-lo.

2.2.4 WFC baseado em grafos

Como o WFC tradicional foi desenvolvido com o intuito de produzir texturas e outros artefatos similares, sua estrutura não permite que os artefatos gerados desviem muito do formato de uma grade retangular ou quadrada. Para permitir artefatos distintos, modificações ao algoritmo ou algum mapeamento de uma grade ao formato desejado é necessária.

Em seu artigo, Kim et al. (2019) propõem uma extensão ao algoritmo WFC para que o mesmo seja capaz de operar em grafos genéricos. A extensão descreve mudanças sobre as estruturas de dados utilizadas e no algoritmo de propagação, que envolvem principalmente a modificação da onda para que a mesma seja representada com um grafo, onde cada célula é mapeada a um nó, e a modificação da estrutura das regras que o algoritmo deve seguir (o que por sua vez acarreta uma mudança no algoritmo de propagação). Em grafos, não há uma inerente relação de direção entre os nós (isto é, não se pode dizer que um nó está "acima" de outro), apenas a relação de adjacência. Assim, os autores decidiram por limitar as regras para que descrevam apenas que peças podem ser adjacentes umas das outras na onda, sem nenhuma especificação de direção. A Figura 12 mostra como as regras são descritas, e um exemplo de geração.

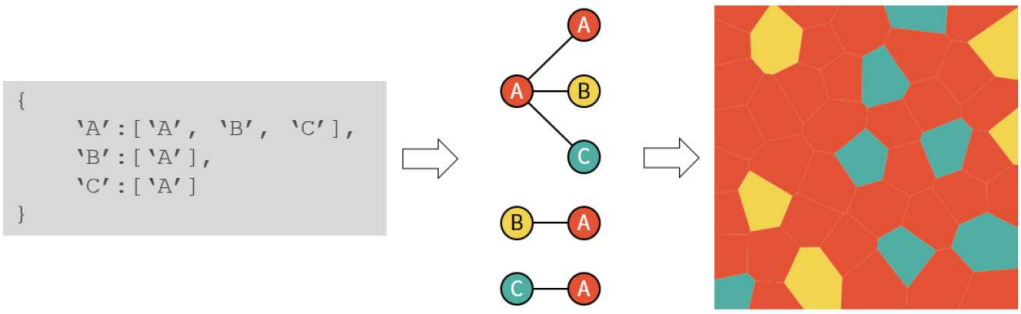


Figura 12 – Regras para colorir um grafo de Voronoi.
Fonte: Extraído de (KIM et al., 2019)

O artigo cita a ocorrência de outro tipo de falha que WFC em grafos genéricos pode ter, na qual a falta de uma forma de verificar direção pode causar o colapso de células em posições que não estão descritas em nenhuma das regras. Isso resulta em peças adjacentes quando não existe uma regra explícita que permita esta adjacência. Este erro, denominado pelo autor como "conflito do propagador", deve gerar *backtracking* ou um reinício global, uma vez que ocorre não como intuito do *designer* que descreveu as regras antes da execução.

Os autores realizaram testes com esta extensão, com o objetivo de validar seu funcionamento. Em particular, um mapa de pontos de interesse em uma ilha tridimensional foi gerado com o uso de regras que descrevem que tipos de pontos (inimigos, moradias,

espaços vazios, entre outros) podem ser encontrados próximos uns dos outros. Apesar disso, nenhum dos exemplos propostos pelos autores verificou se a extensão seria capaz de reproduzir os resultados do algoritmo original; a estrutura do grafo, capaz de representar apenas relações de adjacência e não de direção, pode ter interferido na capacidade de representação da extensão.

3 Desenvolvimento

Neste capítulo, o desenvolvimento da extensão proposta é detalhado. Primeiramente, um conjunto de funcionalidades que devem estar presentes na extensão é definido, as quais permitem o alcance do objetivo principal do trabalho. Um *design* da extensão é então descrito, que elabora adições e modificações nas estruturas de dados e procedimentos do algoritmo WFC para alcançar as funcionalidades desejadas. Por fim, um comparativo das propriedades e características dos métodos é realizado entre o algoritmo WFC original, a extensão proposta neste trabalho, e a extensão baseada em grafos proposta por Kim et al. (2019).

3.1 Funcionalidades Desejadas

Há três principais funcionalidades que deverão ser abordadas pela extensão:

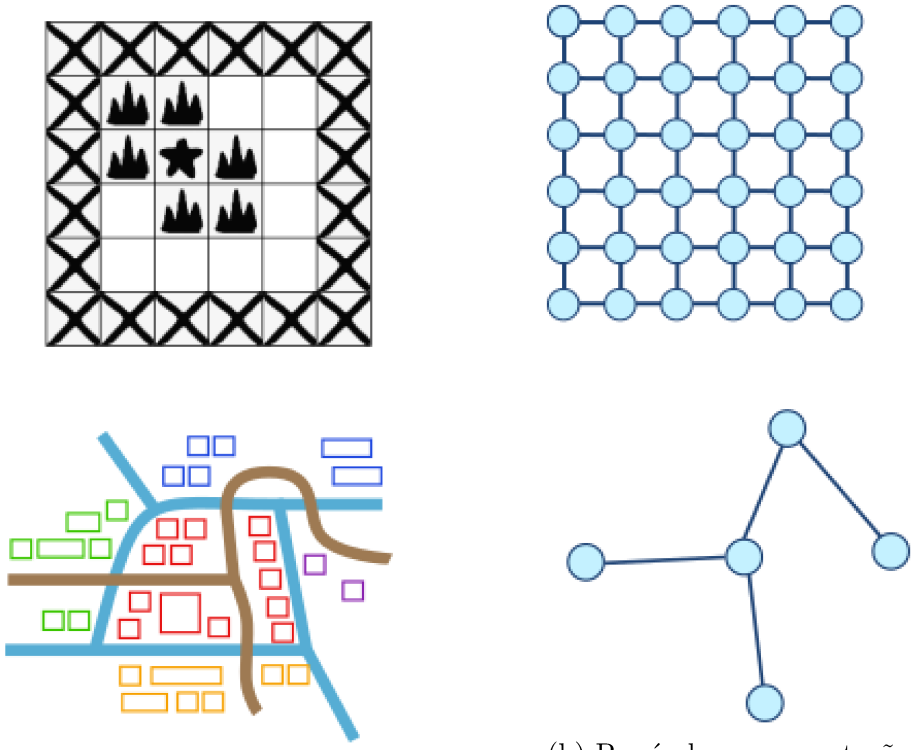
1. A extensão deverá ser capaz de representar e trabalhar com a produção de artefatos em formatos que a forma tradicional do algoritmo WFC não seria capaz de abordar.
2. As modificações feitas pela extensão não deverão impedir que artefatos no formato de grade sejam produzidos.
3. A representação do domínio dos artefatos (isto é, as características de todos os artefatos, como formato, dados adicionais e contexto) na extensão deverá ser padronizada, de tal forma que diferentes domínios poderão ser representados com os mesmos conceitos e estruturas.

As duas primeiras funcionalidades implicam que o algoritmo WFC original será um subconjunto da extensão, uma vez que esta será capaz de produzir os mesmos tipos de artefatos que ele, e isso garantirá um grau de compatibilidade entre os dois.

A terceira funcionalidade tem como objetivo tornar a extensão mais acessível: uma representação comum e flexível para diferentes domínios permitirá que ela seja facilmente adaptável para uso em contextos variados. Porém, como uma característica central do algoritmo WFC é o uso de peças discretas colocadas nas células da onda, existirão domínios, principalmente aqueles que envolvem artefatos formados por dados contínuos, que não podem ser representados pelo WFC sem extensas modificações à lógica do algoritmo. Por isso, no escopo deste trabalho, a padronização das representações na extensão deverá ser descrita em torno de como o algoritmo aborda a geração de artefatos: peças discretas dispostas em células discretas.

3.2 *Design* da Extensão

Para que a extensão seja capaz de abordar uma maior variedade de domínios que o algoritmo original, a onda deve ser representada com alguma estrutura mais flexível. Um grafo é uma opção favorável para isso; a extensão, como o algoritmo original, trabalha com peças discretas que são dispostas em espaços para formar um artefato final, e um grafo permitiria abstrair os locais onde as peças podem ser colocadas na forma de nós (no caso, as células da onda são os nós do grafo), e as direções de propagação entre eles podem ser descritas pelas arestas. Com grafos, o formato ou topologia dos artefatos produzidos poderia então ser mais facilmente manipulada para representar diferentes contextos de uso, como se pode ver na Figura 13.



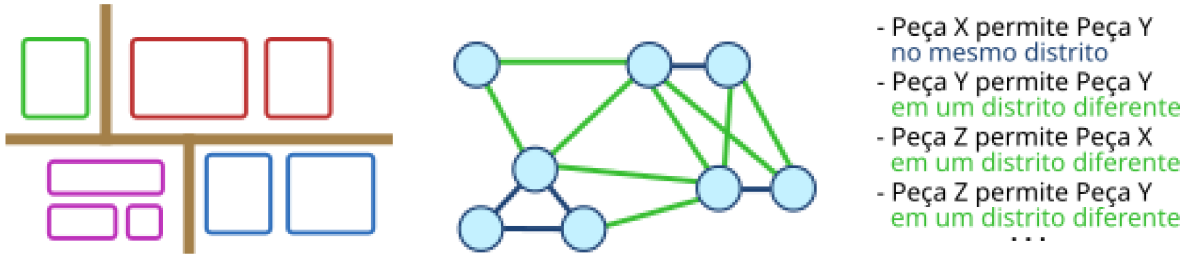
(a) Um tabuleiro de um jogo e um mapa de uma cidade

(b) Possível representação dos domínios na forma de grafo

Figura 13 – Representação de domínios por meio de grafos.

A partir disso, é necessário uma nova forma de representar as regras, uma vez que o algoritmo original depende das direções cardinais para defini-las, e as mesmas não existem em grafos. Para que seja possível descrever com clareza como as células se relacionam, pode-se realizar uma rotulação sobre as arestas do grafo, isto é, associar cada aresta com um valor, denominado “relação” ou “rótulo”, que marca como o par de células adjacentes se relacionam. Por exemplo, duas células em uma geração procedural de um mapa de uma cidade que representam onde construções (as peças neste contexto) podem ser colocadas poderiam ter uma aresta entre si com a relação “distritos diferentes”, e isso poderia ser

utilizado pelas regras para restringir que tipos de construções poderiam existir nestas células. A Figura 14 retrata isso visualmente.



(a) Mapa de exemplo (cada cor representa um distrito diferente)

(b) Possível representação como grafo.

(c) Algumas regras possíveis com os rótulos.

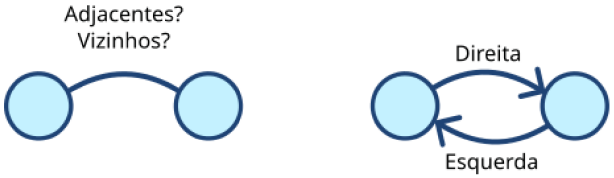
Figura 14 – Rotulação de arestas em um cenário de geração de um mapa de uma cidade. Em 14b, as arestas em verde tem o rótulo "distrito diferente", enquanto as azuis tem o rótulo "mesmo distrito".

Porém, um rótulo único sobre uma aresta impede a representação correta de alguns cenários: casos onde as células não possuem a mesma relação entre si, como em um tabuleiro tradicional, onde, para uma célula c_1 à esquerda de outra c_2, esta está à direita daquela; e casos onde as células podem possuir mais de uma relação pertinente às regras entre si, como no exemplo anterior do mapa da cidade onde duas células, além de poderem estar em distritos diferentes, poderiam estar acima ou abaixo umas das outras. A Figura 15 demonstra visualmente esses casos. Para possibilitar a abordagem dessas situações, a estrutura do grafo foi incrementada com arestas direcionadas, o que permite representar o primeiro caso, e múltiplas arestas por par de nós, o que representa o segundo caso. Dessa forma, o grafo rotulado utilizado pela extensão passa a ser um multigrafo direcionado rotulado.

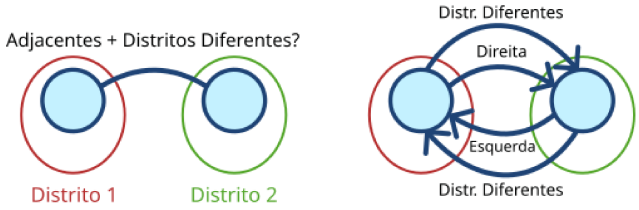
As modificações realizadas na estrutura de dados da extensão acarretam mudanças em como o algoritmo é executado. Primeiramente, o novo formato de regra deve ser definido, e o propagador deve ser modificado para acatar a este formato. Com a ideia de rótulos, podemos definir as regras novamente como triplas: a forma (p_1, p_2, rel) representa que uma peça p_1 em uma célula apenas permite p_2 em outra célula se houver uma aresta com rótulo rel que parte daquela célula a esta. Isto pode ser visto na figura 16. Como as regras ainda são triplas, as mesmas estruturas de dados usadas no algoritmo tradicional poderiam ser usadas para o propagador na extensão.

Com o propagador definido, a lógica da etapa de propagação deve ser modificada para que as mudanças sejam propagadas ao decorrer das arestas de cada célula, e para que o novo formato de regra seja utilizado corretamente. Um pseudocódigo da nova lógica de propagação pode ser visto no Algoritmo 4.

Note que neste algoritmo modificado, ao invés de utilizar as direções, itera-se pelas



(a) Arestas direcionadas permitem descrever a posição relativa entre duas células adjacentes



(b) Múltiplas arestas entre o mesmo par de células permitem descrever mais de uma relação pertinente ao contexto.

Figura 15 – Problemas na rotulação das arestas, e como podem ser resolvidos.

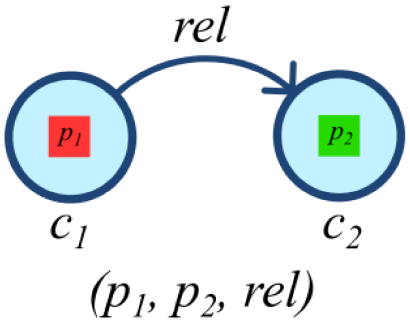


Figura 16 – O novo formato de regra.

células adjacentes à `célulaAtualizada`. Estas células podem ser resgatadas pelas arestas que partem de `célulaAtualizada`, que também é como a relação `rel`, o rótulo de cada aresta, é resgatado na linha 7. A verificação na linha 11 passa a utilizar o novo propagador. Note também que a relação é direcional: a relação na aresta de `célulaAtualizada` à `célulaAdjacente` não é necessariamente a mesma que no sentido contrário.

Por outro lado, no caso da observação, o algoritmo continua o mesmo, uma vez que o cálculo de entropia depende apenas das peças válidas nas células dentro da onda, e não da estrutura da onda em si; o ato de vasculhar as células da onda por aquela com a menor entropia não precisa ser modificado.

Algoritmo 4: Propagação modificada, levando em consideração as relações nas arestas.

Entrada: onda, pilha de propagação, propagador
Saída : onda com as mudanças propagadas, pilha de propagação vazia

```
1  enquanto houverem células na pilha de propagação
2      célulaAtualizada ← remover célula no topo da pilha de propagação (pop)
3      para cada célulaAdjacente adjacente à célulaAtualizada faça
4          se célulaAdjacente já colapsada então
5              continue
6          fim
7          rel ← relação na aresta de célulaAtualizada à célulaAdjacente
8          para cada peça pAdj na lista de célulaAdjacente faça
9              permitida ← false
10             para cada peça pAtu na lista de célulaAtualizada faça
11                 se pAtu permite pAdj dada a relação rel então
12                     permitida ← true
13                     break
14                 fim
15             fim
16             se permitida = false então
17                 remover pAdj da lista de célulaAdjacente
18             fim
19         fim
20         se quantidade de peças na lista de célulaAdjacente mudou então
21             adicionar célulaAdjacente na pilha de propagação
22             marcar célulaAdjacente como colapsada se quantidade de peças na
                lista = 1
23         fim
24     fim
25 fim
26 retornar onda, pilha de propagação
```

3.3 Comparativo

Para realizar um comparativo das características da extensão proposta em relação ao algoritmo WFC original, foram determinados cinco principais pontos de comparação: Domínio, Artefato Gerado, Relações, Formato das Regras e Dados Necessários. A tabela 1 coloca estes pontos em evidência para ambos os algoritmos, e também para a extensão proposta por Kim et al. (2019), que serviu de inspiração para partes da extensão proposta.

Tabela 1 – Tabela comparativa entre características do algoritmo WFC, a extensão baseada em grafos de Kim et al. (2019), e a extensão proposta.

	WFC Tradicional	WFC baseado em grafos	Extensão proposta
Domínio	Grade simples	Grafo	Multigrafo direcionado rotulado
Artefato Gerado	Matriz retangular ou quadrada	Conjunto de nós com valores	Conjunto de nós com valores
Relações	"Cima", "Baixo", "Esquerda" e "Direita"	Adjacente	Relações marcadas nos rótulos da(s) aresta(s)
Formato das Regras	Duas Peças + Direção	Duas Peças	Duas Peças + Relação
Dados Necessários	Tamanho da saída, Peças e Regras	Nós, Arestas, Peças e Regras	Nós, Arestas (com rótulos), Peças, Regras e Relações

Primeiramente, o domínio se refere à estrutura sobre a qual o algoritmo trabalho: o WFC tradicional utiliza uma grade de células (comumente implementada como uma matriz ou vetor); a extensão proposta por Kim et al. utiliza um grafo; e a extensão proposta neste trabalho utiliza, como descrito anteriormente, um multigrafo direcionado rotulado.

O artefato gerado se refere ao formato dos dados gerados pelo algoritmo. O WFC tradicional produz uma matriz retangular ou quadrada que contém o valor colapsado de cada espaço da onda, que pode ser mapeada a um *bitmap*, no caso de texturas, ou usada diretamente para representar mapas bidimensionais. Por outro lado, tanto a extensão proposta por Kim et al. quanto a extensão proposta neste trabalho produzem um conjunto de nós com valores como artefato, onde cada nó é uma célula observada, e os valores associam-nos com as peças nas quais as células colapsaram.

As relações se referem a como cada célula da onda se relaciona umas com as outras. O WFC tradicional, como descrito anteriormente, supõe que cada célula possui células vizinhas acima, abaixo, à esquerda e à direita (exceto as que se encontram nas beiradas e nas quinas da grade), e utiliza essas relações na definição das regras. Na extensão baseada em grafos, cada célula pode possuir N vizinhos com os quais compartilha uma aresta, logo a única relação pertinente é aquela de adjacência. A extensão proposta também utiliza

um grafo para a onda, mas, por meio dos rótulos, as relações entre células podem ser descritas diretamente, e com base no contexto em que o algoritmo é aplicado.

O formato das regras descreve como as regras que o algoritmo utiliza são descritas. O WFC tradicional utiliza regras que podem ser descritas como triplas (p_1, p_2, dir), onde p_1 e p_2 são peças e dir, a direção entre duas células na onda. No WFC de Kim et al., como não há direção e apenas adjacência, as regras seriam dadas apenas por adjacência, ou seja, pares (p_1, p_2). Por fim, na extensão proposta, as regras também são descritas como triplas: (p_1, p_2, rel), onde rel agora é a relação na direção de uma célula à outra.

Por fim, os dados necessários são as informações que o algoritmo necessita de antemão para poder ser executado. No WFC tradicional, é necessário informar as peças, seus pesos (opcionalmente), as regras e o tamanho da saída final, para que a onda seja gerada. No WFC baseado em grafos, é necessário definir os nós e as arestas do grafo que servirá de base para a onda, além das peças e regras de adjacência. Por fim, a extensão proposta exige não só os nós e arestas do grafo para a onda, como também os rótulos das arestas, e o conjunto de relações presentes nesses rótulos, além das peças e regras.

4 Resultados

Neste capítulo, a extensão proposta será validada. Uma implementação de testes será desenvolvida, em conjunto com exemplos que visam verificar se a extensão alcançou as funcionalidades desejadas. Em seguida, discute-se a extensão, os méritos e deméritos observados em decorrer da validação, e suas aplicações em contextos práticos.

4.1 Validação

O funcionamento da extensão foi validado por meio de uma implementação na linguagem de programação C do algoritmo, e de exemplos escritos nas linguagens C e C++ que utilizam a implementação para realizar a produção de conteúdos com domínios distintos. A implementação, juntamente aos exemplos, pode ser encontrada no Repositório GitHub[1].

4.1.1 Reprodução do algoritmo original

Primeiramente, buscou-se implementar exemplos que consigam produzir artefatos no mesmo formato que o algoritmo original do WFC. Isso permitiria verificar se a extensão ainda possui a capacidade de representar e produzir artefatos com topologia de grade, que são comuns em jogos. Os exemplos `wfc` e `wfc_overlap` foram baseados nos exemplos demonstrativos do repositório original de WFC por Gumin (2017), mas utilizam a implementação para a geração. O primeiro gera artefatos como o WFC simples, enquanto o segundo age como o WFC sobreposto.

No exemplo `wfc`, as peças utilizadas na geração são imagens quadradas de $N \times N$ pixeis, que representam cada parte do mapa ou textura final. Estas imagens são agrupadas em *tilesets*, os quais são acompanhados de arquivos XML que descrevem também as regras utilizadas no algoritmo. A leitura das regras, as quais são descritas em um formato especificado por Gumin em seu repositório, foi reimplementada no exemplo. No caso do exemplo `wfc_overlap`, um conjunto de *bitmaps* simples de exemplo é provida, e o pré-processamento para extrair padrões foi reimplementado em C++ com base na implementação original na linguagem C#.

Em ambos os exemplos, a mesma topologia do grafo da onda foi utilizada, que pode ser vista na Figura 17. A topologia visa refletir uma grade, e cada célula um espaço dentro da grade. Assim, para representar o conceito de "direção" na grade, arestas direcionadas são colocadas entre células de espaços que seriam adjacentes na grade, rotuladas

[1] Disponível em: https://github.com/Mateus-Carmo31/wfc-extension.

apropriadamente com uma das quatro direções. Nota-se que, para cada aresta, uma outra no sentido contrário é colocada entre o mesmo par de células, com o rótulo apropriado, para representar a direção oposta e permitir a propagação em ambos os sentidos.

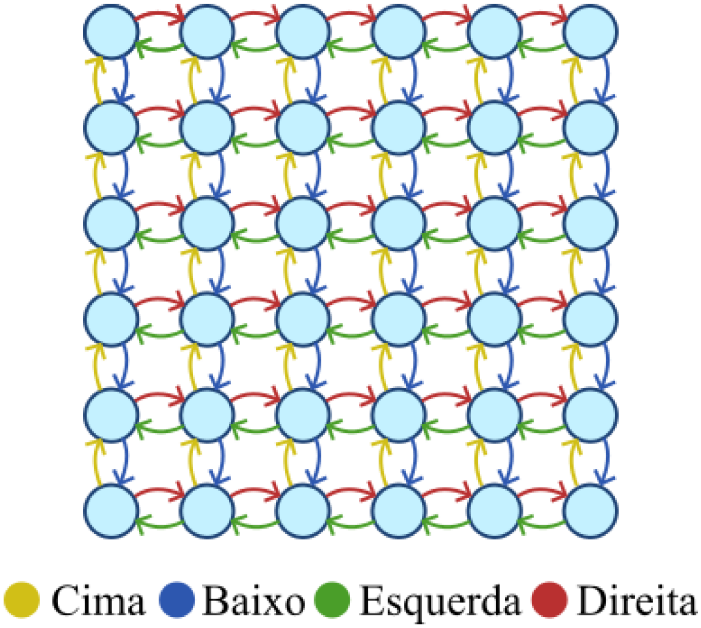
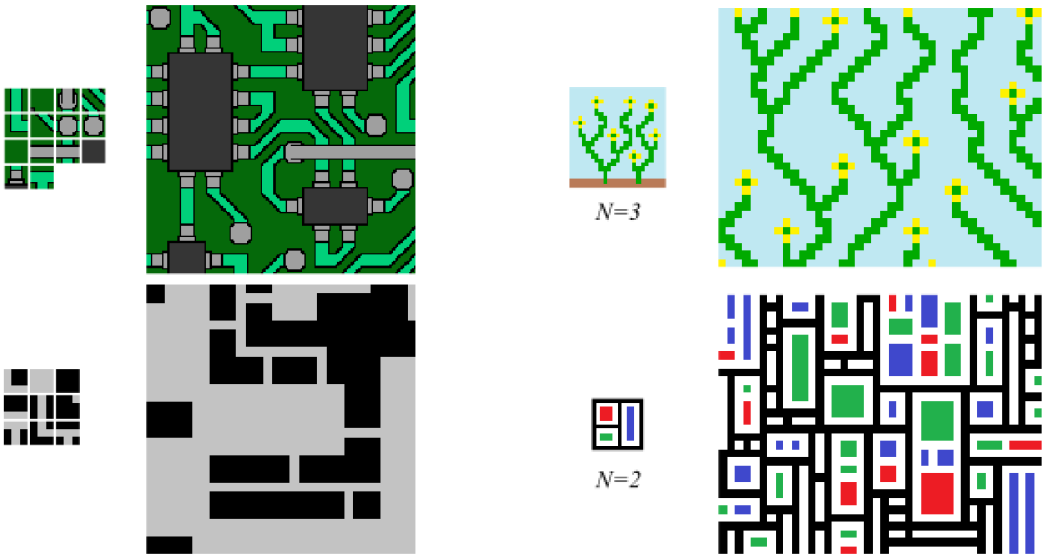


Figura 17 – A topologia da onda nos exemplos `wfc` e `wfc_overlap`, com as arestas coloridas de acordo com o rótulo utilizado.

Com essa topologia, as regras criadas por Gumin em sua implementação original do WFC tradicional puderam ser utilizadas analogamente na extensão. O algoritmo foi capaz de ser executado e produzir tanto mapas, no WFC simples, quanto texturas, no WFC sobreposto. Alguns exemplos produzidos pelo algoritmo podem ser vistos na Figura 18.



(a) `wfc` (b) `wfc_overlap`

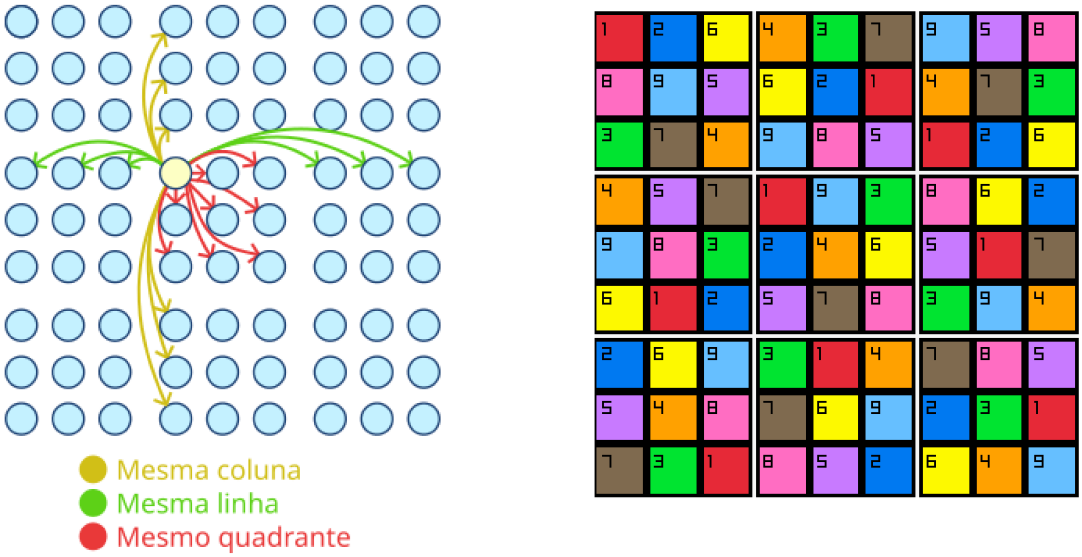
Figura 18 – Artefatos produzidos pelos exemplos `wfc` e `wfc_overlap`.

4.1.2 Exemplos baseados em grafos

Após os primeiros exemplos, buscou-se implementações que testem a capacidade da extensão de representar domínios além de uma grade. Para isso, foram implementados os exemplos `sudoku`, que gera um tabuleiro preenchido do jogo Sudoku, e `graph`, que gera um grafo que representa um mapa de um jogo, com nós preenchidos de acordo com regras, ambos inspirados por cenários abordados em Kim et al. (2019).

Um jogo de Sudoku envolve um tabuleiro similar a uma grade padrão, mas as relações entre cada espaço do tabuleiro não são as mesmas. Uma peça, que no contexto é um número de 1 a 9, colocada em um espaço afeta quais peças podem ser colocadas em cada linha, coluna e quadrante do tabuleiro. Para que a extensão possa ser utilizada nesse contexto, uma mudança da topologia é exigida: ao invés de adicionar arestas entre cada célula com as células imediatamente adjacentes, as arestas devem ser colocadas de tal forma que cada célula seja adjacente de todas as outras células na mesma linha, coluna e quadrante. Os rótulos destas arestas marcam qual destas três relações possíveis entre espaços cada célula possui.

Com essa topologia, as regras a serem utilizadas passam a refletir as próprias regras impostas pelo jogo: cada peça permite qualquer outra peça na mesma linha, coluna ou quadrante, menos ela mesma. A extensão, assim, é capaz de gerar tabuleiros de Sudoku completos, ou até mesmo gerar soluções para tabuleiros parcialmente preenchidos. A Figura 19 mostra a topologia do exemplo, juntamente a um artefato gerado. Em 19a, note que as arestas de apenas uma das célula são mostradas, para evitar dificultar a visibilidade na figura. Na topologia da onda utilizada, todas as células têm arestas similares.



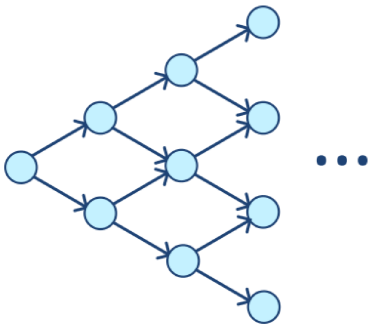
(a) Topologia (b) Artefato produzido

Figura 19 – Topologia para o exemplo `sudoku`, e um artefato produzido.

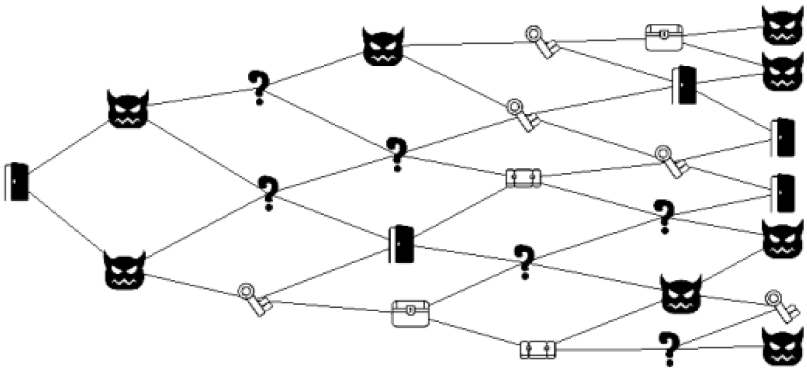
O outro exemplo, `graph`, teve como objetivo explorar um caso onde a topologia

do artefato final também é um grafo. No caso, o exemplo gera uma série de pontos de interesse em um mapa sequencial: cada ponto leva a dois outros pontos em uma série de bifurcações. Os pontos de interesse podem representar combate com inimigos, uma chave para abrir um baú de tesouro, o próprio baú, entre outros tipos de encontros. A topologia utilizada neste exemplo é um grafo, onde cada vértice é um possível ponto de interesse e as arestas indicam diretamente a adjacência dos pontos, logo há um mapeamento quase direto entre o formato do artefato e o formato da onda. As arestas são direcionadas, e partem de cada ponto aos dois pontos adjacentes na camada seguinte.

Para este exemplo, a única relação pertinente entre pontos é se eles são adjacentes ou não. Logo, todas as arestas na onda possuem o mesmo rótulo de "adjacente". As regras, portanto, apenas indicam se um tipo de ponto de interesse permite outro como adjacente. Um exemplo de uma das possíveis malhas de pontos geradas pelo algoritmo, e como a topologia é representada na onda, podem ser vistos na figura 20.



(a) Topologia da onda (pode conter mais colunas)



(b) Artefato gerado

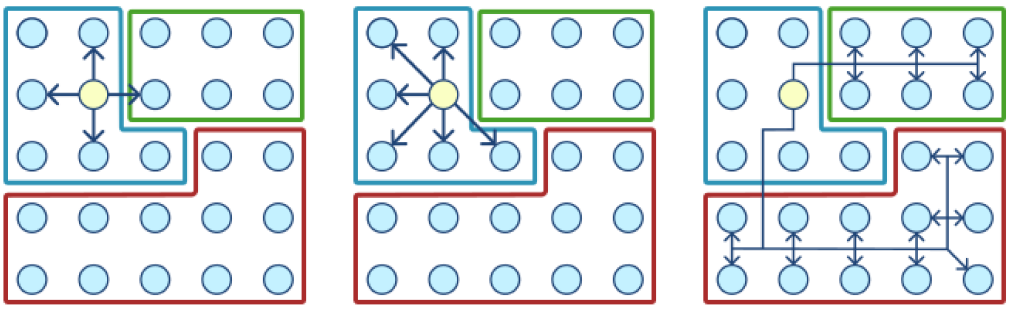
Figura 20 – Topologia do exemplo `graph`, e um exemplo de mapa gerado.

4.1.3 Exemplo prático: geração de um mapa com regiões

No último exemplo implementado, `regions`, buscou-se trabalhar com um cenário prático: a geração de um nível para um jogo de estratégia. Ainda, buscou-se separar o

mapa em "regiões" que podem ser então utilizadas na descrição das regras: o exemplo suportaria regras que impediriam a mesma peça ser colocada em outras regiões após ser colocada em uma, ou impediria uma peça aparecer mais de uma vez na mesma região. Estes tipos de regras descrevem restrições similares à propagação em área descrita por Sandhu, Chen e McCoy (2019), e estendem o controle sobre como o nível é gerado.

A topologia deste exemplo é a mais complexa, mas é baseada nos exemplos `wfc` e `wfc_overlap`. O artefato final é novamente uma grade, portanto as células representam espaços, e as mesmas arestas que indicam direções estão presentes. Porém, um conjunto de regiões é gerado antes da execução do algoritmo, e todos os espaços do mapa são distribuídos entre estas regiões. Quando a topologia é gerada, cada célula de uma região é conectada com todas as outras da mesma região, com arestas marcadas como "mesma região", e também com todas as células de outras regiões, com arestas marcadas como "outra região". Na implementação, as regiões são geradas com um algoritmo de *flood fill*. A Figura 21 demonstra a colocação das arestas, com regiões de exemplo. Apenas as arestas da célula em amarelo são ilustradas, para facilitar a visualização, mas todas as células têm arestas similares.



(a) Arestas com rótulo das direções

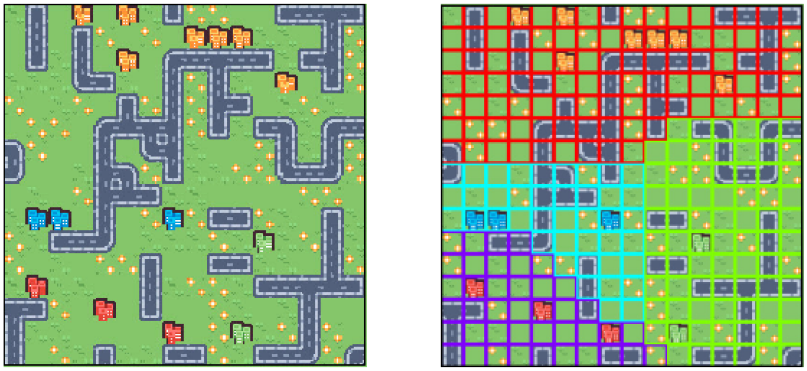
(b) Arestas com rótulo "mesma região"

(c) Arestas com rótulo "outra região"

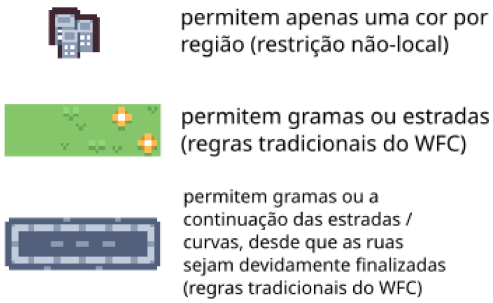
Figura 21 – A topologia na onda do exemplo `regions`, para um mapa 5×5. As arestas de cada rótulo (direções, mesma região e outra região) são mostradas separadamente.

Desta forma, mudanças são propagadas nas quatro direções, mas também para todas as outras células na mesma região e nas outras regiões. Como as arestas relacionadas às regiões possuem rótulos específicos, a informação propagada para toda a região e para as outras regiões será baseada em regras específicas para estas relações, e não atrapalha as regras tradicionais com base nas direções imediatamente adjacentes.

Este exemplo foi implementado utilizando um *tileset* que contém imagens de partes do mapa, e um arquivo TOML que descreve as regras e propriedades de cada peças. Algumas peças foram restritas para aparecer em apenas uma região, e excluir todas as outras do mesmo grupo de peças de aparecer na mesma região. A extensão foi capaz de gerar artefatos que seguem esta restrição, como pode ser visto na Figura 22, que também mostra quais peças possuem quais propriedades.



(a) Mapa gerado (b) Regiões geradas



(c) Lógica das regras

Figura 22 – Um mapa produzido no exemplo `regions`. Apenas uma cor de cidade pode ocorrer por região.

4.2 Discussão

Com a validação, foi possível determinar que o *design* da extensão abrange as funcionalidades desejadas, e é capaz de produzir artefatos por meio delas. A realização de uma implementação de teste, e a implementação dos exemplos, também revelou méritos e deméritos sobre a extensão proposta.

O maior mérito da extensão é que a capacidade de representar e trabalhar com domínios além dos quais o WFC tradicional é capaz, sem a necessidade de adequar a implementação para o domínio desejado, foi alcançada. Os exemplos demonstraram que a extensão consegue de fato gerar artefatos com formatos distintos e, como os exemplos `wfc` e `wfc_overlap` especificamente mostraram, a própria forma original do algoritmo pode ser representada pela extensão: o aumento na capacidade representativa não prejudicou a capacidade da extensão de agir em cenários similares aos onde o algoritmo original é usado.

Porém, a extensão também possui deméritos. O algoritmo estendido e a estrutura de dados escolhida, por sua natureza, exigem mais dados e uma maior quantidade de memória dedicada à execução em comparação com o algoritmo tradicional: é necessário não apenas decidir *a priori* o conjunto de todas as relações que ocorrem entre células da onda, como também é necessário armazenar as arestas e seus rótulos. Ainda, em certos

contextos, a representação de alguns domínios por meio da extensão pode ser complexa e não-intuitiva para um usuário. Isso é aparente no exemplo `regions`, onde a topologia para representar as regiões resulta em uma quantidade consideravelmente grande de arestas entre células que, apesar de serem facilmente configuráveis programaticamente, podem causar dificuldades na depuração de erros.

Em termos das funcionalidades atingidas, vale uma comparação entre a extensão deste trabalho e as propostas de trabalhos relacionados. A extensão proposta por Kim et al. (2019) também tem como objetivo a representação de outros formatos de artefatos, e, como a extensão proposta neste trabalho, faz uso de grafos para alcançar isso. Porém, quando comparada com esta, há uma distinção em termos de descritividade: a extensão de Kim et al. (2019) não é capaz de representar alguns tipos de artefatos e regras, particularmente o algoritmo WFC tradicional, uma vez que as regras e relações que podem ser utilizadas na descrição delas estão limitadas apenas à adjacência das células no grafo. A extensão proposta neste trabalho foi capaz de suprimir esta falta de capacidade descritiva por meio do uso dos rótulos, mas a custo de necessitar de regras mais detalhadas, o que pode tornar a topologia mais complexa, como mencionado anteriormente.

Por outro lado, apesar da extensão proposta possuir uma capacidade de representar restrições mais complexas, ela não é tão flexível neste quesito quanto outras extensões e implementações focadas explicitamente neste aspecto. O trabalho de Sandhu, Chen e McCoy (2019), por exemplo, implementa no algoritmo WFC tradicional a possibilidade de restrições afetarem dinamicamente as células da onda: a observação de uma célula é capaz de provocar observações adicionais em células ao seu redor, à distâncias específicas, sem um preparo prévio da estrutura da onda. Na extensão proposta neste trabalho, tal funcionalidade exigiria uma modificação das arestas do grafo da onda durante a execução do algoritmo, o que poderia provocar instabilidades.

Em prática, esta extensão permite que *designers*, não necessariamente aqueles responsáveis pela implementação do algoritmo, consigam obter um maior controle sobre a geração de conteúdo sem a necessidade de realizar alterações no código interno de execução do algoritmo. Dada uma interface que permita a manipulação da topologia da onda (como adicionar células e arestas), equipes não responsáveis pela implementação ainda poderiam ser capazes de construir geradores de níveis ou outros tipos de artefatos, com a mesma atitude de "caixa-preta" que popularizou o WFC original.

Por exemplo, uma interface poderia ser implementada para um jogo de estratégia, que permitiria a equipe responsável pela criação de níveis facilmente preencher seções dos níveis com elementos de acordo com regras descritas na própria interface. A interface poderia dispor ferramentas para manipulação das peças nas células, similar à interface descrita por Langendam e Bidarra (2022), como também para manipulação da topologia da onda (adição e remoção de células ou arestas). A definição de relações mais complexas

poderia ser abstraída para sistemas de iniciativa mista como esse: topologias complexas como a do exemplo `regions` poderiam ser ofuscadas por ferramentas que geram as arestas e rótulos necessários para o usuário.

5 Conclusão

O algoritmo *WaveFunctionCollapse* se tornou rapidamente um popular algoritmo para a geração procedural de conteúdo em jogos, graças ao seu funcionamento entendível e facilmente controlável para *designers*. Diversas implementações para diferentes plataformas, *frameworks* estão disponíveis com código aberto, e várias versões modificadas oferecem controle adicional para os desenvolvedores que buscam utilizá-las em seus projetos.

O objetivo deste trabalho foi desenvolver uma extensão ao WFC capaz de oferecer aos desenvolvedores que a utilizarem uma maior flexibilidade de uso, e possibilitar a geração de artefatos que o algoritmo original não é capaz de produzir tradicionalmente. Ainda, ao decorrer do desenvolvimento, observou-se a possibilidade de utilizar a flexibilidade da extensão para aumentar o controle sobre os artefatos gerados; por meio da extensão, encontrou-se a possibilidade de representar restrições não-globais, um tipo comum de restrição utilizada em GPC para restringir como o conteúdo é gerado ao decorrer de todo o artefato final, explicitamente dentro da extensão.

Uma implementação de teste foi realizada para a extensão, em conjunto de exemplos que visavam testar as funcionalidades previstas. Por meio desses exemplos, foi verificado que a extensão proposta é capaz produzir artefatos com diversos formatos e restrições distintas, desde aqueles similares aos produzidos pelo WFC tradicional até artefatos com domínios distintos.

Apesar das capacidades da extensão proposta, foram verificados deméritos. O aumento da flexibilidade acarretou um aumento na quantidade de dados que o algoritmo necessita para seu funcionamento, além de aumentar a dificuldade da descrição das restrições utilizadas na geração dos artefatos.

5.1 Trabalhos Futuros

O principal foco deste trabalho foi elaborar o funcionamento geral da extensão e validar se as funcionalidades desejadas foram alcançadas. Por conta disso, o *design* proposto não compara as diferentes estruturas de dados e otimizações que poderiam ser utilizadas. Trabalhos futuros poderiam comparar as melhores formas de representar internamente a estrutura de dados central do algoritmo, o multigrafo direcionado rotulado, e como a escolha poderia afetar a *performance* ou o tempo de resolução do artefato final.

Um outro aspecto que poderia ser explorado seria a capacidade de dinamicamente alterar a estrutura da onda. A capacidade de adicionar células ou arestas em tempo de

execução, com a garantia de que as restrições sejam mantidas e propagadas devidamente, possibilitaria a geração de ainda mais artefatos e outros graus de controle sobre a geração. Trabalhos futuros poderiam implementar estas funcionalidades, e verificar como elas afetam a taxa de contradições e o tempo de execução do algoritmo.

Ainda, o trabalho buscou apenas comparar a extensão proposta ao algoritmo WFC original em termos dos artefatos produzidos, e suas características. Trabalhos futuros poderiam realizar uma comparação mais profunda entre os algoritmos, como comparar o tempo de resolução, o custo de memória, ou o número médio de contradições que ocorrem durante a execução.

Referências

CARLI, D. M. D.; BEVILACQUA, F.; POZZER, C. T.; D'ORNELLAS, M. C. A survey of procedural content generation techniques suitable to game development. In: IEEE. **2011 Brazilian symposium on games and digital entertainment**. [S.l.], 2011. p. 26–35. Citado na página 9.

COMPTON, K. **Practical Procedural Generation for Everyone**. 2017. <https://www.youtube.com/watch?v=WumyfLEa6bU>. Acesso em: 26 de abril de 2024. Citado 2 vezes nas páginas 14 e 16.

GUMIN, M. **WaveFunctionCollapse: Bitmap & tilemap generation from a single example with the help of ideas from quantum mechanics**. 2017. <https://github.com/mxgmn/WaveFunctionCollapse>. Acesso em: 26 de abril de 2024. Citado 4 vezes nas páginas 10, 23, 24 e 36.

HENDRIKX, M.; MEIJER, S.; VELDEN, J. V. D.; IOSUP, A. Procedural content generation for games: A survey. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, ACM New York, NY, USA, v. 9, n. 1, p. 1–22, 2013. Citado 2 vezes nas páginas 15 e 16.

KARTH, I.; SMITH, A. M. Wavefunctioncollapse is constraint solving in the wild. In: **Proceedings of the 12th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2017. p. 1–10. Citado 2 vezes nas páginas 24 e 25.

KIM, H.; LEE, S.; LEE, H.; HAHN, T.; KANG, S. Automatic generation of game content using a graph-based wave function collapse algorithm. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–4. Citado 6 vezes nas páginas 5, 27, 29, 34, 38 e 42.

LANGENDAM, T. S. L.; BIDARRA, R. miwfc-designer empowerment through mixed-initiative wave function collapse. In: **Proceedings of the 17th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2022. p. 1–8. Citado 3 vezes nas páginas 10, 25 e 42.

MATTIOLI, L. R. et al. Uma proposta de um procedimento para a geração semiautomática de ambientes virtuais para subestações de energia elétrica. Universidade Federal de Uberlândia, 2015. Citado na página 9.

SANDHU, A.; CHEN, Z.; MCCOY, J. Enhancing wave function collapse with design-level constraints. In: **Proceedings of the 14th International Conference on the Foundations of Digital Games**. [S.l.: s.n.], 2019. p. 1–9. Citado 4 vezes nas páginas 10, 26, 40 e 42.

SMITH, G. An analog history of procedural content generation. In: BOSTON, MA. **FDG**. [S.l.], 2015. Citado na página 9.

TOGELIUS, J.; YANNAKAKIS, G. N.; STANLEY, K. O.; BROWNE, C. Search-based procedural content generation: A taxonomy and survey. **IEEE Transactions on**

Computational Intelligence and AI in Games, IEEE, v. 3, n. 3, p. 172–186, 2011. Citado na página 15.