UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Rodrigo Torres Meira

AVALIAÇÃO COMPARATIVA EM TERMOS DE MEMÓRIA E TEMPO DE EXECUÇÃO DO ALGORITMO WFC APLICADO À GERAÇÃO DE MAPAS DE JOGOS

CAMPINA GRANDE - PB

2023

Rodrigo Torres Meira

AVALIAÇÃO COMPARATIVA EM TERMOS DE MEMÓRIA E TEMPO DE EXECUÇÃO DO ALGORITMO WFC APLICADO À GERAÇÃO DE MAPAS DE JOGOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Herman Martins Gomes

CAMPINA GRANDE - PB

2023

Rodrigo Torres Meira

AVALIAÇÃO COMPARATIVA EM TERMOS DE MEMÓRIA E TEMPO DE EXECUÇÃO DO ALGORITMO WFC APLICADO À GERAÇÃO DE MAPAS DE JOGOS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Herman Martins Gomes

Orientador – UASC/CEEI/UFCG

Andrey Elisio Monteiro Brito

Examinador – UASC/CEEI/UFCG

Melina Mongiovi Sabino

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 17 de NOVEMBRO de 2023.

CAMPINA GRANDE - PB

RESUMO

O algoritmo *Wave Function Collapse* (WFC) desempenha um papel de destaque na indústria de jogos, sendo comumente utilizado para a geração automática de texturas e mapas. Tal utilização faz parte da área de geração procedural de conteúdo (*Procedural Content Generation* - PCG), a qual tem como motivações aprimorar a rejogabilidade e aliviar a carga de trabalho dos desenvolvedores na criação manual de conteúdo. O uso de PCG tem experimentado um notável aumento nos últimos anos, impulsionado pelo constante crescimento no tamanho e na complexidade dos jogos produzidos.

Embora diversas técnicas de PCG já tenham sido amplamente documentadas e testadas, o WFC é uma abordagem relativamente recente que carece de avaliações abrangentes quanto à sua eficácia em comparação com outras técnicas. Este trabalho de conclusão de curso tem como objetivo preencher essa lacuna, realizando uma análise da complexidade do algoritmo WFC em termos de tempo e recursos de hardware, comparando-o com alternativas no contexto da geração de mapas de jogos.

Foram executados algoritmos de PCG em mapas de diversos tamanhos, variando de 100px a 1000px. Durante as execuções, foram registrados os dados de consumo de memória e tempo de execução. Os resultados demonstraram que, entre os algoritmos avaliados, o WFC se destaca no quesito consumo de memória, superando os demais algoritmos nesse aspecto. Já o *Binary Space Partitioning Room Placement (*BSPRP) demonstrou ser o mais eficiente em termos de tempo, superando significativamente o desempenho do WFC. Por fim, o *Random Room Placement* (RRP) se mostrou o menos eficiente, tanto em consumo de memória quanto em tempo de execução.

COMPARATIVE EVALUATION IN TERMS OF MEMORY AND EXECUTION TIME OF THE WFC ALGORITHM APPLIED TO GAME MAP GENERATION

ABSTRACT

The *Wave Function Collapse* (WFC) algorithm plays a prominent role in the gaming industry, commonly used for the automatic generation of textures and maps. Such usage is part of the field of *Procedural Content Generation* (PCG), which aims to enhance replayability and alleviate the workload of developers in manual content creation. The use of PCG has experienced a notable increase in recent years, driven by the constant growth in the size and complexity of produced games.

Although several PCG techniques have already been extensively documented and tested, WFC is a relatively recent approach that lacks comprehensive evaluations of its effectiveness compared to other techniques. This thesis aims to fill this gap by conducting an analysis of the complexity of the WFC algorithm in terms of time and hardware resources, comparing it to alternatives in the context of game map generation.

PCG algorithms were executed on maps of various sizes, ranging from 100px to 1000px. During the executions, memory consumption and execution time data were recorded. The results showed that, among the evaluated algorithms, WFC excels in terms of memory consumption, surpassing the other algorithms in this aspect. On the other hand, the *Binary Space Partitioning Room Placement* (BSPRP) proved to be the most efficient in terms of time, significantly outperforming the performance of WFC. Finally, the *Random Room Placement* (RRP) was the least efficient, both in terms of memory consumption and execution time.

AVALIAÇÃO COMPARATIVA EM TERMOS DE MEMÓRIA E TEMPO DE EXECUÇÃO DO ALGORITMO WFC APLICADO À GERAÇÃO DE MAPAS DE JOGOS

Rodrigo Torres Meira
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
rodrigo.meira@ccc.ufcg.ed.br

Herman Martins Gomes
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
hmg@computacao.ufcg.edu.br

RESUMO

O algoritmo Wave Function Collapse (WFC) desempenha um papel de destaque na indústria de jogos, sendo comumente utilizado para a geração automática de texturas e mapas. Tal utilização faz parte da área de geração procedural de conteúdo (Procedural Content Generation - PCG), a qual tem como motivações aprimorar a rejogabilidade e aliviar a carga de trabalho dos desenvolvedores na criação manual de conteúdo. O uso de PCG experimenta um notável aumento nos últimos anos, impulsionado pelo constante crescimento no tamanho e na complexidade dos jogos produzidos.

Embora diversas técnicas de PCG já tenham sido amplamente documentadas e testadas, o WFC é uma abordagem relativamente recente que carece de avaliações abrangentes quanto à sua eficácia em comparação com outras técnicas. Este trabalho de conclusão de curso tem como objetivo preencher essa lacuna, realizando uma análise da complexidade do algoritmo WFC em termos de tempo e recursos de hardware, comparando-o com alternativas no contexto da geração de mapas de jogos.

Foram executados algoritmos de PCG em mapas de diversos tamanhos, variando de 100px a 1000px. Durante as execuções, foram registrados os dados de consumo de memória e tempo de execução. Os resultados demonstraram que, entre os algoritmos avaliados, o WFC se destaca no quesito consumo de memória, superando os demais algoritmos nesse aspecto. Já o Binary Space Partitioning Room Placement (BSPRP) demonstrou ser o mais eficiente em termos de tempo, superando significativamente o desempenho do WFC. Por fim, o Random Room Placement (RRP) se mostrou o menos eficiente, tanto em consumo de memória quanto em tempo de execução.

Keywords

Comparação de algoritmos, conteúdo gerado proceduralmente, análise de eficiência, *autoencoder*, WFC, BSPRP, RRP.

1. INTRODUÇÃO

No cenário atual de desenvolvimento de jogos, algoritmos de Geração Procedural de Conteúdo (*Procedural Content Generation* referido nesse documento como PCG) desempenham um papel fundamental [1] na geração de níveis, terrenos e até texturas [2], permitindo a criação de ambientes dinâmicos e ricos em detalhes. Nos anos recentes, as empresas de jogos vêm buscando criar jogos cada vez maiores, tanto em duração quanto em complexidade, aumentando o uso e importância desses tipos de ferramentas [1].

O uso de PCG no desenvolvimento de jogos está se tornando cada vez mais prevalente, com o objetivo de aprimorar a jogabilidade, a reprodutibilidade e a geração de conteúdo ao longo do desenvolvimento. As técnicas de PCG são empregadas de formas diversas em diferentes gêneros de jogos, com algumas áreas, como a geração de níveis e a instanciação de entidades, sendo particularmente propensas a utilizar PCG em comparação com geração de quebra-cabeças, enredos e sistemas dinâmicos[1].

A família de algoritmos de PCG envolve diversos tipos de algoritmos diferentes [10], dos quais os mais comumente utilizados para criação de mapas são *algoritmos espaciais*, que funcionam manipulando o espaço para gerar conteúdo. A saída é criada utilizando uma entrada com estrutura, como uma grade. Outro grupo de algoritmos fundamenta-se em restrição, buscando, a partir de um estado inicial, chegar a um estado final, seguindo regras previamente definidas [10]. Foco deste trabalho de conclusão, o algoritmo *Wave Function Collapse* (WFC) [3] pertence ao segundo grupo, considerado mais complexo, e será comparado com os mais eficientes do primeiro grupo [9].

O algoritmo WFC foi proposto em 2016 por Maxim Gumin. O algoritmo recorre a uma técnica de resolução por restrições para extrapolar quantidades maiores de conteúdos a partir de uma pequena amostra de entrada, encontrando padrões de conjunto de pixels, e tentando reconstruí-los na saída final, uma definição mais detalhada do algoritmo está na próxima seção.

Infelizmente, o WFC em seu estado normal é altamente complexo e de baixa eficiência. Para poder alcançar um nível de comparação adequado, foi realizada a combinação do algoritmo WFC com uma arquitetura de rede neural chamada *AutoEncoder*, como proposto em [3]. O uso do *AutoEncoder* não altera a essência do funcionamento do algoritmo, mas permite que ele gere saídas maiores graças à capacidade de operar sobre dimensões mais abstratas na entrada.

O estudo apresentado no presente trabalho tem como objetivo comparar o algoritmo WFC com os algoritmos *Binary Space Partitioning Room Placement* (BSPRP) e o *Random Room Placement* (RRP)[6], os quais serão definidos na seção seguinte. Eles se destacaram como os mais eficientes entre os algoritmos de Geração de Conteúdo Procedural (PCG) para a criação de mapas 2D em jogos [9]. A comparação envolverá a execução desses três algoritmos, utilizando diferentes tamanhos de mapas, e avaliando as diferenças no tempo de execução e uso de recursos. A pesquisa visa identificar se o WFC supera ou é superado por esses algoritmos em termos de eficiência.

Existe outro trabalho produzido com o objetivo de analisar algoritmos PCG, [9], porém devido à data em que foi produzido, antes do WFC se tornar popular, ele só analisa a eficiência de alguns *algoritmos espaciais*.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Algoritmos

Nesta seção será definido e explicado em mais detalhes os algoritmos *Wave Function Collapse (WFC), Random Room Placement (RRP), Binary Space Partitioning Room Placement (BSPRP)*, em função de seus papéis de destaque em pesquisas anteriores [9].

Também será apresentada uma explicação mais detalhada quanto ao uso do AutoEncoder em conjunto com o WFC.

2.1.1 Wave Function Collapse Clássico

O algoritmo Wave Function Collapse (WFC) é uma técnica de geração de conteúdo procedural que ganhou destaque por sua capacidade de criar texturas e mapas complexos de forma automatizada. A abordagem subjacente ao WFC se assemelha à mecânica quântica, onde as "funções de onda" representam a probabilidade de diferentes estados do sistema. No contexto do WFC, os "tiles" (peças) representam partes do conteúdo a ser gerado, como pixels de textura ou células de um mapa. O algoritmo começa com um grid vazio, e a partir de um conjunto de "tiles" de entrada e algumas restrições, ele infere e preenche o grid de maneira a respeitar essas restrições, criando um resultado que é coerente e esteticamente agradável.

Originalmente criado para geração de textura, esse algoritmo ganhou popularidade no uso na criação de mapas procedurais [4, 5]. Por exemplo, o jogo *Bad North* [4] utiliza uma versão do algoritmo para gerar as ilhas existentes nos seus mapas, utilizando heurísticas das vizinhanças entre tipos de ilhas e regras de construção internas das ilhas.

Esse processo pode ser observado nas figuras 1 e 2, demonstrando a transformação do exemplo de entrada na saída com o tamanho desejado.

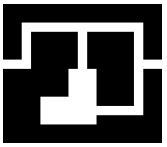

Figura 1. Exemplo de input

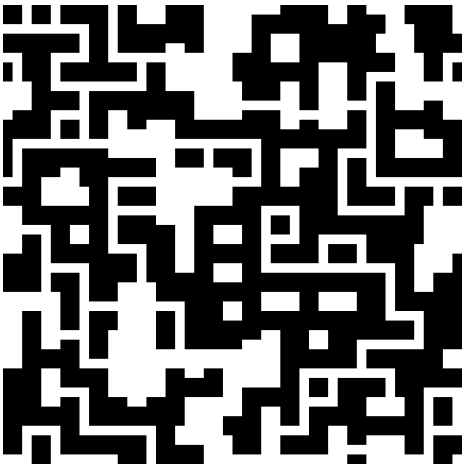

Figura 2. Exemplo de saída

2.1.2 Random Room Placement

O algoritmo Random Room Placement (RRP) é um método de geração de conteúdo procedural, da família de *algoritmos espaciais*, utilizado na criação de ambientes em jogos e simulações. Segue uma abordagem de força bruta para posicionar quartos de forma aleatória em um mapa. O processo começa com a geração de um quarto com dimensões aleatórias, onde a variabilidade do tamanho do quarto é controlada por valores máximo e mínimo predefinidos. Em seguida, o algoritmo escolhe um ponto aleatório no mapa para servir como o canto inferior esquerdo do quarto, assegurando que esse ponto esteja distante o suficiente das bordas superior e direita do quarto para garantir tamanho adequado. A validação de que o quarto não intersecta com outro quarto existente no mapa é realizada, e, se houver interseção, o algoritmo tenta gerar um novo ponto de posicionamento. Esse processo é repetido até que o número desejado de quartos tenha sido colocado no mapa ou até que um limite máximo de tentativas seja alcançado.

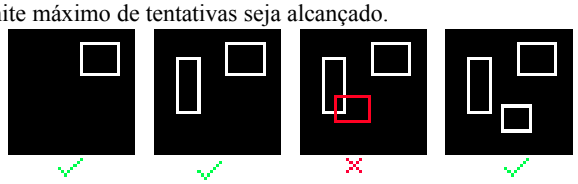

Figura 3. Exemplo de execução do RRP

Após isso, um algoritmo qualquer de geração de corredores é então utilizado para ligar os quartos e finalizar o mapa. Fundamentando-se nos resultados obtidos em [9] será utilizado o algoritmo BSP Corridors para produção desses corredores.

2.1.3 Binary Space Partitioning Room Placement

O algoritmo *Binary Space Partitioning Room Placement* (BSPRP) é uma técnica mais avançada de geração de conteúdo procedural, também da família de *algoritmos espaciais*. Tem como princípio fundamental a divisão binária do espaço do mapa para colocar quartos de maneira eficiente e estruturada. O processo começa dividindo o espaço do mapa em duas áreas, geralmente mediante uma linha vertical ou horizontal, criando dois subespaços. Essa divisão é repetida recursivamente, subdividindo-se cada subespaço até que um critério de parada seja atingido, como o tamanho mínimo dos quartos. Quando a subdivisão é concluída, os quartos são posicionados em cada subespaço, e os corredores são criados para conectar esses quartos, resultando em uma configuração mais regular do ambiente.

Figura 4. Exemplo de execução do BSPRP

O algoritmo BSPRP oferece a vantagem de criar espaços com organização satisfatória e com um layout equilibrado, o que pode melhorar a jogabilidade e a estética dos jogos. Além disso, ele permite um controle refinado sobre o tamanho e a posição dos quartos, tornando-o uma escolha popular para jogos que requerem mapas complexos e bem definidos.

A partir dos resultados obtidos em [9] será utilizado o algoritmo BSP Corridors para produção dos corredores.

2.2 Arquitetura *AutoEncoder*

Autoencoders (AE) são redes neurais que visam copiar suas entradas para suas saídas. Eles trabalham compactando a entrada em uma representação de espaço latente e, em seguida, reconstruindo a saída dessa representação [7].

O *AutoEncoder* pode ser dividido entre dois componentes: o *Encoder*, responsável por compactar a entrada para o espaço

latente, e o *Decoder*, responsável por descompactar do espaço latente para as dimensões de entrada.

2.2.1 *Wave Function Collapse com AutoEncoder*

Devido à definição do funcionamento do WFC, baseado em resolução por restrição, ele possui baixa escalabilidade, sua complexidade aumentando de forma quase exponencial em relação à quantidade de restrições [3]. Para combater esse problema, foi proposto o uso de um *AutoEncoder*, simples e completamente conectado, para extrair vetores latentes que possam representar abstrações de agrupamentos semelhantes de pixels [3], obtendo-se uma redução de 28×28 para apenas 5×5 pixels.

Incluiremos essa variante, fundamentada no *autoencoder* sugerido por [3], no estudo comparativo. Dessa forma, será possível avaliar um algoritmo que opera sobre entradas e saídas com dimensões menores que os demais algoritmos.

2.2.2 *Uso do AutoEncoder*

Diferentemente dos demais algoritmos, o WFC opera buscando e usando padrões existentes em suas entradas, dessa forma conseguindo tirar proveito do espaço latente gerado pelos *AutoEncoder* e, assim, poder encontrar padrões nesse espaço latente produzido, funcionando como mais uma camada de abstração. Ao utilizar a saída do *Encoder* como entrada para o algoritmo WFC, ele consegue gerar uma saída adequada, no espaço latente, mas com uma dimensionalidade maior. Dessa forma, o *Decoder* possui mais dados para reconstruir, e a saída final passa a possuir um tamanho maior.

Os demais algoritmos não trabalham com esse reconhecimento de padrões e reutilização deles, visto que não recebem exemplos de entrada, apenas informações como tamanho do quarto e número de quartos a serem gerados.

3. METODOLOGIA

Para a realização do experimento comparativo, será necessária a geração dos mapas e a definição dos parâmetros de tamanho e quantidade de quartos dos mapas. Para verificar os resultados serão escolhidos os parâmetros de eficiência dos tempos e uso de recursos.

Cada um dos algoritmos definidos anteriormente será posto para gerar mapas seguindo os parâmetros definidos nesta seção. A maioria dos parâmetros fundamenta-se nos experimentos realizados por [9], a fim de manter os resultados dos algoritmos testados por eles similares aos nossos.

3.1 Parâmetros dos mapas

3.1.1 *Tamanho dos mapas*

Tomando como base os experimentos realizados por [9], os algoritmos irão gerar mapas de tamanhos 100×100 até 1000×1000, com incrementos de 100px, isso se faz necessário para ser possível a comparação entre os resultados dos experimentos. Para cada tamanho serão gerados 20 mapas. Para medir a eficiência dos algoritmos serão comparadas os resultados das médias do tempo utilizado para gerar cada tamanho dos mapas.

3.1.2 *Quantidade de quartos*

Para esse experimento não será definido um número máximo de quartos, apenas um valor mínimo de quartos, cuja escolha do valor mínimo irá variar conforme o tamanho de cada mapa estudado. Inspirado pelo experimento de [9], a quantidade mínima de quartos será dependente do tamanho do mapa e buscando-se ter um quarto para cada 400px² no mapa, de modo que em um mapa de 100px de lado tenha aproximadamente 20 quartos. Assim, com base na fórmula abaixo, obtém-se a quantidade mínima de quartos a ser utilizada:

$$Q = M \times 0,15\%$$

Onde Q é a quantidade mínima de quartos e M o tamanho do mapa. Tendo como exemplo o menor tamanho de mapa de 10000 px² (100×100 px), obtêm-se como resultado a quantidade mínima de 15 quartos

3.2 Parâmetros de comparação

3.2.1 *Eficiência de Tempo*

Para avaliar o tempo de execução dos algoritmos será utilizado como parâmetro a média de tempo medido para gerar cada um dos tamanhos de mapas definidos anteriormente, além disso, para melhorar a percepção da eficiência dos resultados, serão comparados: a relação de aumento de tempo com o aumento do tamanho do mapa, para que possíveis usuários destes algoritmos possam ter melhor conhecimento sobre a escalabilidade e uso específico de cada um dos algoritmos.

Esses parâmetros foram escolhidos pelo impacto que eles têm quando usados pelos desenvolvedores para agilizar a produção de um jogo, visto que não é desejável gastar horas esperando o algoritmo terminar a sua execução para poder utilizar os resultados dele. Além disso, da perspectiva do jogador, quanto mais tempo ele espera para o carregamento do mapa, maiores são as chances dele perder a motivação de jogá-lo.

3.2.2 *Uso de Recursos*

Além das medidas de tempo, será acompanhado o uso de memória de cada um dos algoritmos, através da biblioteca padrão do *python tracemalloc*[1]. Os resultados encontrados serão comparados para cada tamanho de mapa e em relação a sua escalabilidade.

Esse parâmetro é importante para a otimização de jogos, por permitir que os desenvolvedores conheçam o consumo de memória que essas ferramentas vão ocupar, de forma que eles possam alocar essa memória onde seja mais prescindível.

4. RESULTADOS

Os experimentos foram executados segundo as definições da seção anterior, onde cada um dos algoritmos foi executado 20 vezes com cada tamanho de mapa diferente, indo de 100×100 a 1000×1000 pixels. Durante o experimento, a complexidade de espaço foi medida usando a média dos espaços relatados pela biblioteca *tracemalloc*. Vale ressaltar que o WFC com *AutoEncoder* acaba trabalhando com uma dimensionalidade diferente, mas, como queremos verificar o resultado final, o uso do *AutoEncoder* fez com que a atuação do WFC fosse numa dimensionalidade bem menor. Para gerar o resultado final de 1000×1000px, o WFC só precisou usar uma grade de 33×33px, utilizando o *Decoder* para transformar a dimensionalidade final em 1000×1000px.

Com o propósito de otimizar a compreensão dos resultados, as tabelas nesta seção foram sintetizadas para incluir apenas as informações mais relevantes à análise. As tabelas completas, abrangendo os resultados de cada tamanho de mapa individual, estão disponíveis de forma mais detalhada no apêndice.

4.1 A importância do AutoEncoder

Em seu estado natural, o WFC possui alta complexidade, o que normalmente tornaria essa comparação desnecessária, mas devido à sua capacidade de guardar abstrações mais altas de informações, podendo mudar formatos de quartos, espessura de corredores e de paredes e criando conexões mais diversas. Ele pode ser usado em conjunto com um AutoEncoder para poder extrair da entrada inicial padrões mais abstratos, e o WFC ser usado nesses padrões, que, por sua vez, são "traduzidos" pelo Decoder.

Inicialmente esse trabalho realizou a comparação entre o WFC clássico com os demais algoritmos, porém isso rapidamente se

[1] *https://docs.python.org/3.11/library/tracemalloc.html*

mostrou infrutífero, visto que uma execução do WFC com 200×200px levou aproximadamente meia hora, como pode ser visto na Tabela 4. Cada execução de 60 segundos precisava ser executada pelo menos 20 vezes, e o mapa 300×300px chegou a levar mais de um dia. Porém, com a introdução do AutoEncoder, o WFC só precisa trabalhar com a geração de grades 33×33px para a saída final alcançar 1000×1000px, resultando em tempo e complexidade comparáveis àqueles dos demais algoritmos comparados.

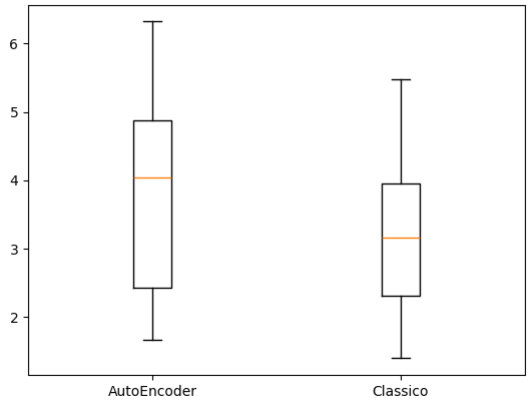

Figura 5. Comparação de tempo (segundos) entre versões diferentes do algoritmo WFC

A diferença entre o WFC clássico e sua utilização com o AutoEncoder pode ser vista na figura 5, onde ambos os algoritmos são executados em grids 100×100px. É importante ressaltar que o resultado final com AutoEncoder representa uma área de 3225×3225px.
É possível observar a semelhança de tempo entre o modelo clássico e o modelo com o AutoEncoder, ambos levando aproximadamente 6 segundos (em média, após 20 execuções), porém o resultado final do WFC AE possui 3325px enquanto o WFC Clássico contém apenas 100px.
Portanto, devido a restrições de tempo e comparações desnecessárias, o WFC clássico só foi executado com baixas dimensionalidades (de 100px² e 200px²). Os resultados apresentados a seguir são todos com base no tamanho final gerado pelos algoritmos.

4.2 Tempo de execução

4.2.1 RRP

Tamanho do mapa	Aumento % do tamanho do mapa	Tempo	Aumento % de tempo
100px	-	0,117 ms	-
100px - 200px	100%	0,4406 ms	376,58%
100px - 500px	400%	7,1963 ms	6150,69%
100px - 1000px	900%	102,4736 ms	87584,3%
500px - 1000px	100%	95,2773 ms	1302,79%

Tabela 1. Resultados de tempo do RRP

É possível observar na Tabela 1 e Figura 6 que a taxa de crescimento do tempo é maior que o aumento da dimensionalidade dos mapas. Além disso, é evidente também que a diferença da taxa de crescimento entre o mapa e o tempo aumenta quanto maior a dimensionalidade. Quando o mapa vai de 100px para 200px, um crescimento de 100%, o tempo sofre um aumento de 376,58%, 3,7658 vezes o aumento de tamanho. Já ao se observar o aumento de 500px para 1000px, outro aumento de 100%, a duração cresce em uma taxa de 1302,79%, 13,0279 vezes maior. O aumento do quanto ele aumentou de dimensionalidade baixa para alta foi de 376,58% para 1302,79%, um aumento de 3,46 vezes.

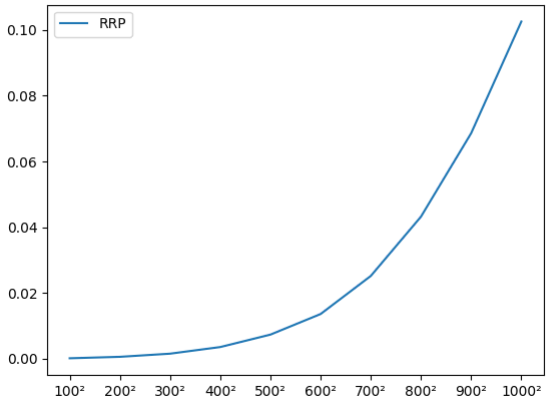

Figura 6. Curva de resultados de tempo (em segundos) do algoritmo RRP em função da área em pixels

4.2.2 BSPRP

Tamanho do mapa	Aumento % do tamanho do mapa	Tempo	Aumento % de tempo
100px	-	0,0474 ms	-
100px - 200px	100%	0,0751 ms	158,44%
100px - 500px	400%	1,8375 ms	3876,59%
100px - 1000px	900%	5,3048 ms	11191,57%
500px - 1000px	100%	3,4673 ms	183,95%

Tabela 2. Resultados de tempo do BSPRP

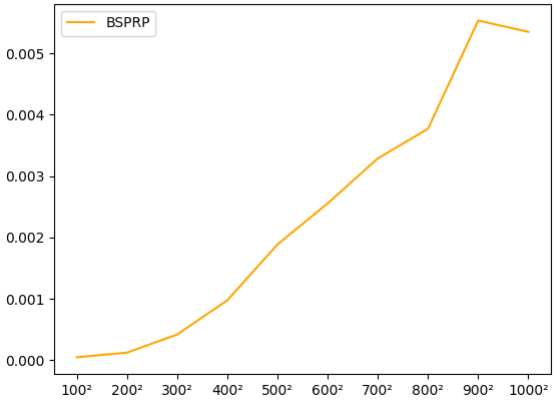

Figura 7. Curva de resultados de tempo (em segundos) do algoritmo BSPRP em função da área em pixels

Ao analisar o BSPRP é possível observar, na Tabela 2 e Figura 7, que, enquanto a taxa de crescimento do tempo é maior que o aumento do mapa, ele cresce com uma proporção muitas vezes menor que o do *RRP*, que a diferença de crescimento entre dimensionalidades é muitas vezes menor. O aumento de 100px para 200px impacta no tempo em uma taxa de crescimento de 158,44%, enquanto de 500px para 1000px o aumento é apenas 183,95%, um crescimento de apenas 1,16 vezes.

4.2.3 WFC

Tamanho do mapa	Aumento % do tamanho do mapa	Tempo	Aumento % de tempo
100px	-	0,96 ms	-
100px - 200px	100%	0,09 ms	9,37%
100px - 500px	400%	7,49 ms	780,21%
100px - 1000px	900%	59,49 ms	6196,87%
500px - 1000px	100%	52,0 ms	615,38%

Tabela 3. Resultados de tempo do WFC AE

Como pode ser visto na Tabela 3, o aumento de tempo do algoritmo WFC pode ser visto bem similar ao do RRP, enquanto o valor dele de 500px para 1000px é menor que o do RRP, 615,38% do WFC e 1302,79% do RRP. O aumento dele em questão da dimensionalidade é muito maior, de 9,37% para 615,38% (65,67 vezes maior), comparado ao do RRP de apenas 3,46 vezes. Juntamente com a Figura 8, isto indica que rapidamente o WFC ultrapassaria o RRP se as dimensões continuassem aumentando.

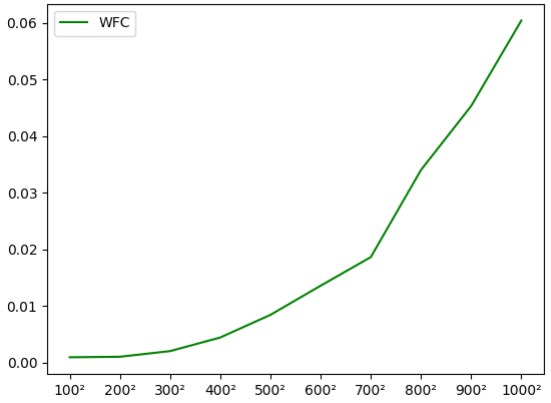

Figura 8. Curva de resultados de tempo (em segundos) do algoritmo WFC AE em função da área em pixels

4.2.4 WFC Clássico

Tamanho do mapa	Aumento % do mapa	Média de Tempo registrado	Aumento em relação ao anterior	Aumento % de tempo
100px	-	3,242 s	-	-
200px	100%	52,919 s	49,677 s	1532,313 3%

Tabela 4. Resultados de tempo do WFC Clássico

O WFC clássico é muito inferior aos demais, enquanto todos os outros são medidos em milissegundos, esse algoritmo teve que ser medido em segundos, com o seu menor tamanho superando o tempo do WFC AE em mais de 3000 (3 mil) vezes. E seu aumento de 100px para 200px, um aumento de 100% do mapa, impactou em um aumento de 1532,31% no tempo. Vale ressaltar que esse aumento em dimensionalidade pequena é equivalente ao maior aumento de qualquer um dos algoritmos anteriores.

4.3 Consumo de Memória

4.3.1 RRP

Tamanho do mapa	Aumento % do tamanho do mapa	Memória	Aumento % de memória
100px	-	9096 KiB	-
100px - 200px	100%	4625 KiB	50,85%
100px - 500px	400%	72532 KiB	797,41%
100px - 1000px	900%	317368 KiB	3489,09%
500px - 1000px	100%	244836 KiB	299,94%

Tabela 5. Resultados de consumo de memória do RRP

É possível observar, na Tabela 5 e pela Figura 9, que a diferença na taxa de crescimento entre o mapa e o consumo de memória aumenta quanto maior a dimensionalidade. Quando o mapa vai de 100px para 200px, um aumento de 100%, o consumo sofre um aumento de 50,85%. Já ao se observar o aumento de 500px para 1000px, outro aumento de 100%, a duração aumenta em 299,94%. O aumento do quanto ele aumentou de dimensionalidade baixa para alta foi de 50,85% para 299,94%, um aumento de 5,90 vezes. É interessante notar que o aumento foi maior do que o da memória, levando a uma curva mais acentuada, tornando-o mais ineficiente para dimensões mais altas.

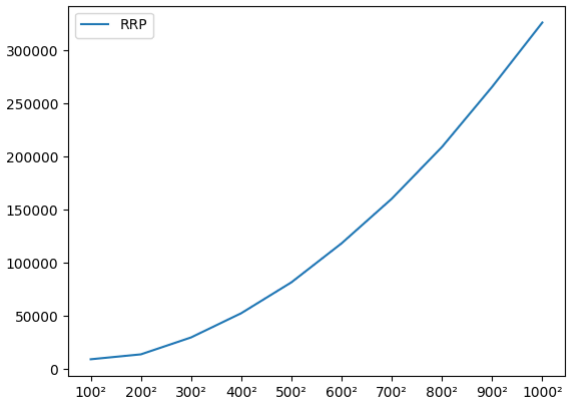

Figura 9. Curva de resultados de consumo de memória (em KiB) do algoritmo RRP em função da área em pixels

4.3.2 *BSPRP*

Tamanho do mapa	Aumento % do tamanho do mapa	Memória	Aumento % de memória
100px	-	9792 KiB	-
100px - 200px	100%	5600 KiB	57,19%
100px - 500px	400%	67724 KiB	691,63%
100px - 1000px	900%	250568 KiB	2558,91%
500px - 1000px	100%	182844 KiB	235,88%

Tabela 6. Resultados de consumo de memória do BSPRP

Diferente do aumento de tempo, é possível notar na Tabela 6 e Figura 10 que o aumento da memória está bem maior que o do aumento do mapa, indicando um aumento muito grande para mapas muito maiores.
Enquanto no aumento de tempo ele foi de 158,44% (100px - 200px) para 183,95% (500px para 1000px), um aumento de 1,16 vezes, o aumento de memória foi de 57,19% (100px - 200px) para 253,88% (500px para 1000px), um aumento de 4,44 vezes. Claramente uma escalabilidade bem menor que a de tempo.

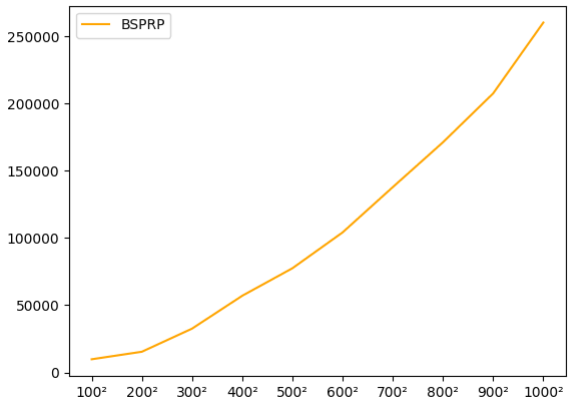

Figura 10. Curva de resultados de consumo de memória (em KiB) do algoritmo BSPRP em função da área em pixels

4.3.3 *WFC*

Tamanho do mapa	Aumento % do tamanho do mapa	Memória	Aumento % de memória
100px	-	4352,64 KiB	-
100px - 200px	100%	9,8 KiB	0,23%
100px - 500px	400%	107,2 KiB	2,46%
100px - 1000px	900%	348,0 KiB	8,0%
500px - 1000px	100%	240,8 KiB	5,0%

Tabela 7. Resultados de consumo de memória do WFC AE

Observando a Tabela 7, nota-se que a memória do WFC consegue se manter com um aumento muito baixo. Isso se deve ao fato de que o maior consumo dele é decorrente dos padrões que ele encontra, e não do tamanho final do mapa, somado isso com o fato de usar o AutoEncoder, o consumo fica grandemente reduzido. Porém, pode ser notar que o aumento dos aumentos de memória cresce numa proporção também muito grande, como a do tempo, ele vai de 0,23% para 5,0%, um aumento de 21,74 vezes, isso pode indicar que o consumo de memória ultrapassaria eventualmente os dos demais algoritmos, dado um aumento de mapa suficiente.

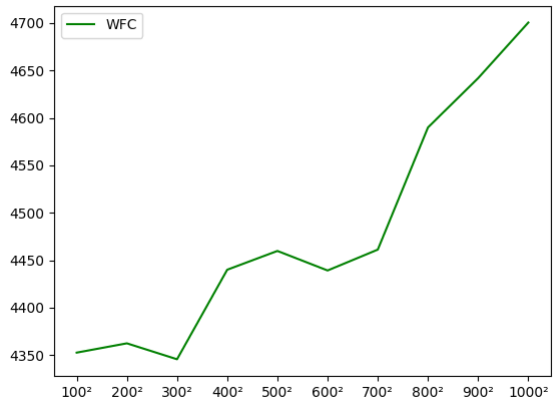

Figura 11. Curva de resultados de consumo de memória (em KiB) do algoritmo WFC AE em função da área em pixels

4.3.4 WFC Clássico

Tamanho do mapa	Aumento % do mapa	Média de memória registrado	Aumento em relação ao inicial	Aumento % de memória
100px	-	7088,6 KiB	-	-
200px	100%	15223 KiB	8134 KiB	114,75%

Tabela 8. Resultados de consumo de memória do WFC Clássico

É possível ver o aumento considerável que o WFC clássico sofre logo em dimensões baixas, já no primeiro aumento de dimensão, já alcançando mais de 114,75% de aumento de consumo de memória. Porém, é interessante observar que mesmo assim o consumo de memória dele ainda é menor que o RRP e o BSPRP, porém com forte indício que rapidamente superaria eles se rodado em dimensões mais altas.

5. ANÁLISE DE RESULTADOS

5.1 Tempo de Execução

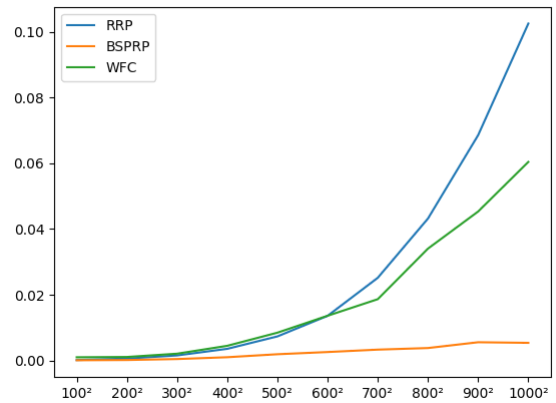

Figura 12. Comparação de curvas dos resultados de tempo (em segundos) em função da área em pixels

Quando comparados um com o outro, é possível observar que o mais eficiente em relação ao tempo é o BSPRP. Por utilizar uma divisão binária no mapa, ele consegue gerar as divisões da forma mais rápida e direta dentre os algoritmos.

Já o WFC é consideravelmente mais lento que o BSPRP, mas consegue ainda ser um pouco mais eficiente que o RRP. Isso pode ser atribuído, em parte, ao fato de que o RRP, em algumas situações, pode inadvertidamente levar a uma sala em uma posição altamente desfavorável, o que pode impactar negativamente o cálculo de novas salas. Quando outras salas entram em conflito com a que está em uma posição desfavorável, o algoritmo se vê obrigado a regenerar as salas repetidamente na busca por uma solução adequada.

Vale ressaltar que apesar de, no escopo analisado, o RRP aparentar apresentar um comportamento mais exponencial que o WFC, podemos observar que pela taxa de crescimento encontrada nos experimentos e em dimensões mais altas, o WFC apresenta indicações de superar eventualmente o RRP.

Graças ao uso do *AutoEncoder*, o WFC apresenta resultados competitivos de tempo, mas novamente vale ressaltar que esse comportamento não é escalável para dimensões mais elevadas. Porém, o tamanho de 1000px escolhido para esse experimento é maior do que o normalmente utilizado para a maioria dos jogos, então essa escalabilidade ainda é válida.

5.2 Uso de Memória

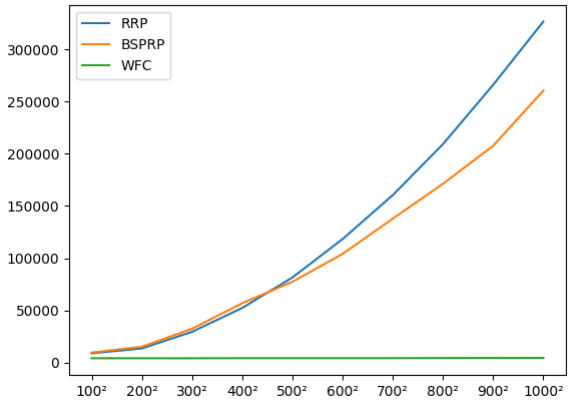

Figura 13. Comparação de curvas dos resultados de consumo de memória (em KiB) em função da área em pixels

Em contraste com os resultados relacionados ao tempo de execução, o WFC demonstrou um desempenho notável no que se refere ao consumo de memória.

Enquanto os outros algoritmos apresentam um aumento diretamente proporcional ao tamanho do mapa, por precisarem guardar as salas individualmente para poder gerar as conexões entre elas, o WFC guarda apenas os padrões encontrados no input. Dessa forma, o tamanho resultante final possui menor impacto na diferença do consumo de memória do algoritmo.

6. CONCLUSÃO

Esse estudo foi realizado com o objetivo de comparar 3 algoritmos de geração de mapas em jogos, o *Wave Function Collapse* (WFC), *Random Room Placement* (RRP), e *Binary Space Partitioning Room Placement* (BSPRP). Os resultados proporcionaram uma análise abrangente do desempenho desses algoritmos em termos de tempo de execução e uso de recursos, trazendo informações importantes sobre o desempenho relativo desses algoritmos e as implicações de escalabilidade em cenários de jogos.

No que diz respeito ao tempo de execução, o estudo demonstrou que o BSPRP é o mais eficiente, superando significativamente os outros algoritmos. Ele se beneficia de uma divisão binária direta do espaço do mapa, permitindo gerações mais rápidas. O WFC com *AutoEncoder*, embora mais lento do que o BSPRP, ainda supera o RRP, que pode ser impactado negativamente por salas mal posicionadas.

No entanto, a análise indicou que, em dimensões mais altas, o WFC AE pode eventualmente superar o RRP em termos de tempo de execução, pois sua taxa de aumento apresenta indícios de ser maior que a do RRP. Além disso, a introdução do *AutoEncoder* no WFC demonstrou que essa abordagem pode competir em termos de tempo, desde que o tamanho do mapa não seja excessivamente grande. No entanto, é importante notar que o tamanho do mapa de 1000px é mais do que suficiente para a maioria dos jogos, tornando essa escalabilidade aceitável.

Em relação ao uso de memória, o WFC se destacou positivamente. Enquanto os outros algoritmos demonstraram um aumento proporcional no consumo de memória à medida que o tamanho do mapa aumentava, o WFC manteve um aumento muito menor, devido à sua capacidade de armazenar padrões em vez de salas individuais. No entanto, como observado, o aumento na diferença de consumo de memória no WFC indica que ele pode eventualmente superar o RRP e o BSPRP à medida que as dimensões do mapa aumentam.

Vale ressaltar que embora a eficiência do BSPRP seja melhor que o do WFC, os mapas gerados pelo WFC tendem a ser de maior qualidade, pela sua capacidade de utilizar padrões latentes das entradas, dessa forma criando mapas mais diversos [2].

Em resumo, os resultados deste estudo oferecem informações valiosas para desenvolvedores de jogos que buscam escolher algoritmos de geração de mapas adequados para suas necessidades. O BSPRP é recomendado para gerações de mapas rápidos, enquanto o WFC, especialmente com o uso do AutoEncoder, pode ser uma alternativa viável para mapas de tamanhos moderados. No entanto, deve-se ter em mente que o uso de memória do WFC aumenta significativamente com o aumento das dimensões do mapa, tornando-o menos eficiente em cenários de alta escala. Portanto, a escolha do algoritmo dependerá das especificidades do projeto de desenvolvimento do jogo, dependendo se será priorizado a qualidade dos mapas ou a eficiência da geração.

6.1 Trabalhos Futuros

Há várias direções promissoras para trabalhos futuros que podem aprimorar a compreensão e a aplicação dos algoritmos estudados. Realizar uma avaliação qualitativa mais aprofundada é uma direção promissora, uma análise critica das saídas geradas por cada algoritmo e suas capacidades em lidar com casos extremos e complexos permitirá uma compreensão mais aprofundada relação entre eficiência e qualidade dos resultados.

Além disso, com a popularização do WFC, é interessante ficar atento com possíveis variações dele que emerjam nos próximos anos, melhorando sua eficiência e tornando sua escolha em projetos ainda mais viável.

7. REFERÊNCIAS

[1] Dahrén, Martin, “The Usage of PCG Techniques Within Different Game Genres.” Disponível em: <https://www.diva-portal.org/smash/get/diva2:1604550/FULLTEXT02>. Acesso em 19 de junho de 2023.

[2] G. Smith, E. Gan, A. Othenin-Girard, and J. Whitehead, “PCG-based game design: enabling new play experiences through procedural content generation.”. Disponível em: <https://www.researchgate.net/publication/229020641_PCG-based_game_design_Enabling_new_play_experiences_through_procedural_content_generation>. Acesso em 19 de junho de 2023.

[3] Isaac Karth and Adam M. Smith, “WaveFunctionCollapse: Content Generation via Constraint Solving and Machine Learning”. Disponível em: <https://adamsmith.as/papers/tog-wfc.pdf>. Acesso em 19 de junho de 2023.

[4] O. Stålberg, R. Meredith, and M. Kvale, “Bad North,” Plausible Concept.

[5] J. Grinblat and C. B. Bucklew, “Caves of Qud,” 2010.

[6] Baron, Jessica. (2017). Procedural Dungeon Generation Analysis and Adaptation. 168-171. 10.1145/3077286.3077566.

[7] Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

[8] Stuart Russell, Peter Norvig, “Artificial Intelligence: A Modern Approach (4th Edition)”. Pearson 2020, ISBN 9780134610993

[9] Z. Monaghan, “Comparing Procedural Content Generation Algorithms for Creating Levels in Video Games,” Technological University Dublin, School of Computing, 2019.

[10] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 9, 1, Article 1 (February 2013), 22 pages. https://doi.org/10.1145/2422956.2422957

Sobre o Autor:

Rodrigo Torres Meira é aluno do curso de Ciência da

Computação na Universidade Federal de Campina Grande.

Apêndice

Tamanho do mapa	Aumento % do mapa	Média de Tempo registrado	Aumento em relação ao inicial	Aumento % de tempo
100px	-	0,1169 ms	-	-
200px	100%	0,5576 ms	0,4406 ms	376,58%
300px	50%	1,5137 ms	0,9561 ms	171,46%
400px	33,33%	3,5304 ms	2,0167 ms	133,23%
500px	25%	7,3132 ms	3,7827 ms	107,14%
600px	20%	13,5805 ms	6,2672 ms	85,69%
700px	16,66%	25,1635 ms	11,5830 ms	85,29%
800px	14,28%	43,2114 ms	18,0478 ms	71,72%
900px	12,5%	68,6068 ms	25,3954 ms	58,77%
1000px	11,11%	102,5905 ms	33,9836 ms	49,53%

Tabela 9, Resultados de consumo de tempo do RRP

Tamanho do mapa	Aumento % do mapa	Média de memória registrado	Aumento em relação ao inicial	Aumento % de memória
100px	-	9096 KiB	-	-
200px	100%	13721 KiB	4625 KiB	50,84%
300px	50%	29671 KiB	15950 KiB	116,24%
400px	33,33%	52500 KiB	22829 KiB	76,94%
500px	25%	81628 KiB	29128 KiB	55,481%
600px	20%	118372 KiB	36744 KiB	45,013%
700px	16,66%	160424 KiB	42052 KiB	35,52%
800px	14,28%	209208 KiB	48784 KiB	30,409%
900px	12,5%	265900 KiB	56692 KiB	27,09%
1000px	11,11%	326464 KiB	60564 KiB	22,77%

Tabela 10. Resultados de consumo de memória do RRP

Tamanho do mapa	Aumento % do mapa	Média de Tempo registrado	Aumento em relação ao anterior	Aumento % de tempo
100px	-	0,04739995 ms	-	-
200px	100%	0,1225 ms	0,07510004797950387	158,43907530777702%
300px	50%	0,41809998 ms	0,29559998074546456	241,30610386907506%
400px	33,33%	0,97470003 ms	0,556600047275424	133,12606337576884%
500px	25%	1,88490003 ms	0,9102000040002167	93,38257684178 39%
600px	20%	2,55550002 ms	0,6705999840050936	35,5774827362671%
700px	16,66%	3,28869995 ms	0,7331999368034303	28,6910558320682%
800px	14,28%	3,77180002 ms	0,48310006968677044	14,68969733945127%
900px	12,5%	5,53620001 ms	1,7643999890424311	46,7787257497724%
1000px	11,11%	5,35220001 ms	-0,18400000408291817	-3,32357941639507%

Tabela 11. Resultados de consumo de tempo do BSPRP

Tamanho do mapa	Aumento % do mapa	Média de memória registrado	Aumento em relação ao inicial	Aumento % de memória
100px	-	9792 KiB	-	-
200px	100%	15392 KiB	5600 KiB	57,18954248366012
300px	50%	32592 KiB	17200 KiB	111,74636174636174
400px	33,33%	57040 KiB	24448 KiB	75,01227295041727
500px	25%	77516 KiB	20476 KiB	35,8976157082749
600px	20%	104196 KiB	26680 KiB	34,41870065534857
700px	16,66%	137892 KiB	33696 KiB	32,33905332258436
800px	14,28%	171260 KiB	33368 KiB	24,19864821744552
900px	12,5%	207628 KiB	36368 KiB	21,235548289151
1000px	11,11%	260360 KiB	52732 KiB	25,39734525208546

Tabela 12. Resultados de consumo de memória do BSPRP

Tamanho do mapa	Aumento % do mapa	Média de Tempo registrado	Aumento em relação ao anterior	Aumento % de tempo
100px	-	0,96 ms	-	-
200px	100%	1,05 ms	0,08999999999999986	09,37499999999997%
300px	50%	2,05 ms	1,0000000000000007	95,2380952380953%
400px	33,33%	4,45 ms	2,4000000000000004	117,07317073170728%
500px	25%	8,05 ms	3,5999999999999996	80,89887640449436%
600px	20%	13,6 ms	5,550000000000001	68,944099378882%
700px	16,66%	18,65 ms	5,050000000000004	37,1323529411765%
800px	14,28%	34,05 ms	15,400000000000006	82,57372654155498%
900px	12,5%	45,35 ms	11,300000000000004	33,18649045521292%
1000px	11,11%	60,45 ms	15,099999999999987	33,2965821389195%

Tabela 13. Resultados de consumo de tempo do WFC AE

Tamanho do mapa	Aumento % do mapa	Média de memória registrado	Aumento em relação ao inicial	Aumento % de memória
100px	-	4352,6 KiB	-	-
200px	100%	4362,4 KiB	9,8 KiB	0,2252%
300px	50%	4345,6 KiB	-16,8 KiB	-0,3851%
400px	33,33%	4440 KiB	69,4 KiB	2,1723%
500px	25%	4459,8 KiB	19,8 KiB	0,4459%
600px	20%	4439,2 KiB	-20,6 KiB	-0,4619%
700px	16,66%	4461,2 KiB	22 KiB	0,4956%
800px	14,28%	4590 KiB	128 KiB	2,8871%
900px	12,5%	4642 KiB	-52 KiB	1,1329%
1000px	11,11%	4700,6 KiB	58,6 KiB	1,2624%

Tabela 14. Resultados de consumo de memória do WFC AE