Geração procedural de níveis de jogos utilizando autômatos celulares

Arthur Toyokawa Sperandio, Eduardo Henrique Molina da Cruz

[1]Instituto Federal do Paraná (IFPR) - Campus Paranavaí
Rua José Felipe Tequinha, 1400 – Jardim das Nações — Paranavaí – PR – Brasil

arthur.toyokawa@gmail.com, eduardo.cruz@ifpr.edu.br

Abstract. *Procedural content generation (PCG) is a mechanism used in digital games to provide a wide variety of content to players at a low development cost. One form of this content is maps composed of terrains and populated with additional elements. Cellular autômata (CA) is a data processing technique in which objects in a matrix change their state based on a set of transition rules. This work aims to analyze how both techniques can be used together for map generation. A software was implemented with the proposed techniques, and an analysis of the characteristics of the generated maps was conducted.*

Resumo. *A geração procedural de conteúdo (GPC) é um mecanismo utilizado em jogos digitais para fornecer uma variedade grande de conteúdo ao jogador com baixo custo de desenvolvimento. Uma dessas formas de conteúdo são mapas compostos por terrenos e populados com outros elementos. Autômatos celulares (AC) é uma técnica de processamento de dados em que objetos de uma matriz mudam seu estado com base em um conjunto de regras de transição. Este trabalho tem como objetivo analisar como ambas técnicas podem ser utilizadas em conjunto para a geração de mapas. Um software foi implementado com as técnicas propostas e foi feita uma análise das características dos seus mapas gerados.*

1. Introdução

A geração procedural de conteúdo (GPC) se refere à criação algorítmica de algum produto final. A GPC é utilizada por muitos jogos digitais para fornecer aos jogadores um fluxo contínuo de experiências e interações novas. Ela pode ser utilizada para a construção de ambientes virtuais, a criação de texturas, tabuleiros, música e diversos outros tipos de conteúdo [Oliveira et al. 2024].

A principal razão para usar GPC é a eliminação da mão de obra humana (designer ou artista) para a criação de conteúdo. Desde que os jogos de computador foram inventados, o número de horas de trabalho requeridas para o desenvolvimento de um jogo comercial bem-sucedido aumentou constantemente. Hoje em dia, é comum que um jogo seja desenvolvido por centenas de pessoas ao longo de um período de um ano ou mais. Isso leva a uma situação em que menos jogos são lucrativos e menos desenvolvedores podem criar um jogo. Uma empresa de desenvolvimento de jogos que conseguisse substituir alguns dos artistas e designers por algoritmos teria uma vantagem competitiva, já que os jogos poderiam ser produzidos de forma mais rápida e econômica, mantendo a qualidade [Shaker et al. 2016].

Autômatos celulares (AC) são ferramentas versáteis capazes de representar quase todos os sistemas evolutivos existentes. Suas principais características são a computação descentralizada, onde cada célula evolui com base apenas nos estados anteriores do sistema ao seu redor. Atualmente, pesquisadores de diversas áreas utilizam modelos de autômatos celulares para simular diferentes tipos de aplicações na biologia evolutiva, na dinâmica das reações químicas, nos sistemas dinâmicos da física, no comportamento de mercados entre outros [Castro and de Oliveira Castro 2008].

Na geração procedural de mapas com autômatos celulares, o AC é aplicado a uma matriz caótica, organizando as células e produzindo um resultado final com características mais "orgânicas" em comparação com outros algoritmos de GPC. Como exemplo, [Johnson et al. 2010] gerou um algoritmo para a geração de mapas de cavernas infinitas e demonstrou seu baixo custo computacional. [Minini 2021] utilizou para a geração de florestas e para adicionar terrenos em mapas existentes ou suavizar partes de mapas gerados com outros algoritmos.

Esta pesquisa tem como objetivo analisar técnicas de geração procedural e autômatos celulares, bem como a forma como são combinados para a criação de mapas para jogos. A metodologia adotada envolveu uma investigação no funcionamento da geração procedural de mapas, identificando as qualidades desejadas em um gerador, as técnicas mais comuns e suas características. Também foi abordado o conceito de autômatos celulares e os passos necessários para a criação de mapas por meio deles. Foram comparados trabalhos que aplicam autômatos celulares na geração procedural, analisando as diferenças entre suas aplicações e usos. Por fim, foi desenvolvido um software para a geração de mapas utilizando as técnicas estudadas, avaliando as características dos mapas gerados e o impacto de diferentes parâmetros de geração sobre os mapas finais.

Este artigo está organizado da seguinte forma. A Seção 2 detalha o funcionamento de autômatos celulares, seu uso dentro de geração procedural, e como algoritmos podem ser modificados para obter mapas com características diferentes. A Seção 3 examina pesquisas que utilizam autômatos celulares para geração de mapas, analisando suas metodologias, aplicações e resultados. A Seção 4 analisa o software implementado, abordando os algoritmos e metodologias utilizados para a geração de mapas, por fim mostrando os passos implementados para gerar um mapa final. A Seção 5 aborda a avaliação experimental dos mapas gerados pelo software, comparando características de resultados utilizando diferentes parâmetros. A Seção 6 apresenta as conclusões acerca dos dados analisados e a implementação, bem como trabalhos futuros.

2. Fundamentação teórica

Nesta seção, será analisado o funcionamento de autômatos celulares em GPC, o funcionamento do gerador, sua ordem de execução, qual escolhas foram feitas sobre seu funcionamento e exemplos de possíveis alternativas e seus resultados.

2.1. Autômatos celulares

Autômato celular é uma forma de computar dados utilizada em múltiplas aplicações como física e biologia, além de programação. Ele funciona dentro de uma matriz de células, com cada célula tendo um estado dentro de vários estados possíveis. O processamento é feito em gerações, em que cada célula analisa seus vizinhos e pode mudar o seu estado

baseado nos valores desses vizinhos [Castro and de Oliveira Castro 2008]. Um algoritmo básico de autômatos celulares pode ser definido como:

1. Cria uma matriz e a popula com células.
2. Começa uma geração.
   (a) É criado uma copia da matriz para guardar os resultados.
   (b) Cada célula corre o algoritmo de autômatos gerando um novo estado que é guardado na matriz de resultados.
   (c) A matriz de resultados sobrescreve a matriz inicial.
3. As gerações são repetidas por um número pré-definido de vezes ou até que uma outra condição é alcançada.

2.1.1. Conway's Game of Life

De acordo com [Bays 2010] Conway's Game of Life é um dos algoritmos de autômato celular mais famosos. Ele funciona com células com os estados 0 (morto) e 1 (vivo). A cada geração, as células verificam as 4 células ao redor (acima, abaixo, direita e esquerda) e mudam o seu estado segundo as regras:

- Se é uma célula viva e tem 2 ou menos células vivas vizinhas morre por falta de população.
- Se é uma célula viva e tem 4 células vivas vizinhas morre por superpopulação.
- Se é uma célula morta e tem 3 células vivas vizinhas nasce por reprodução.
- Se a célula não se encaixa em nenhuma das regras acima o seu estado é mantido para a próxima geração.

2.1.2. Autômatos celulares em geração procedural

Para o uso em geração de mapas procedurais, são utilizados geralmente dois estados, representando terreno transitável e terreno intransitável. Esses estados são utilizados para gerar terrenos mais irregulares, geralmente com o objetivo de simular formações naturais como cavernas, ilhas ou florestas.

2.1.3. Estado inicial

Como estado inicial da matriz de células, é gerada uma matriz com valores aleatórios chamada de *noise grid*. A proporção de terrenos impassíveis para passáveis afeta fortemente o resultado final; quanto maior o número de paredes, mais bloqueios e áreas inacessíveis são geradas.

2.1.4. Análise de vizinhança

Cada célula verifica o estado de suas células vizinhas. A sua vizinhança pode ter múltiplas definições, sendo as mais comuns a de Von Neumann, que conta as quatro adjacentes da célula: abaixo, acima, à esquerda e à direita, e a de Moore, que conta, além das 4 adjacentes, as 4 diagonais: esquerda acima, esquerda abaixo, direita acima e direita abaixo.

Cada célula, então, executa algoritmos denominados por [Ashlock and Kreitzer 2020] de *fashion*, em que cada célula tende a seguir os valores de sua vizinhança de uma maneira similar a como estilos de roupa se espalham em um grupo de pessoas. Isso tende a agrupar as células em conjuntos de terreno impassáveis e passáveis, que podem depois ser separados em salas e populados com conteúdo, entre outros tratamentos.

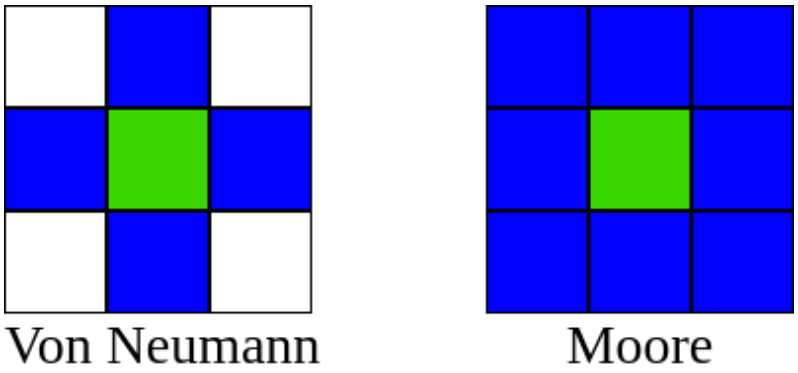


Figura 1. Vizinhanças de autômatos celulares.

2.2. Funcionamento de um gerador de mapas

O gerador segue vários passos que afetam o resultado final e podem ser alterados dependendo do mapa requerido.

2.2.1. Gerando a grid inicial

Para a criação do estado inicial, foi preenchida uma matriz de células com valores de 0 ou 1 aleatoriamente. A chance de uma célula ser uma parede pode ser aumentada ou diminuída dependendo das características desejadas do mapa; quanto maior a chance de popular paredes, menor o tamanho médio de salas. Na Figura 2, são mostrados três mapas finalizados com única diferença sendo o estado inicial, onde a chance de células serem paredes é 50, 45, e 40% respectivamente. Como exemplo, um mapa com mais paredes pode ser utilizado para representar um arquipélago de pequenas ilhas cercadas pelo mar; um mapa com menos paredes pode significar uma área mais aberta como uma floresta.

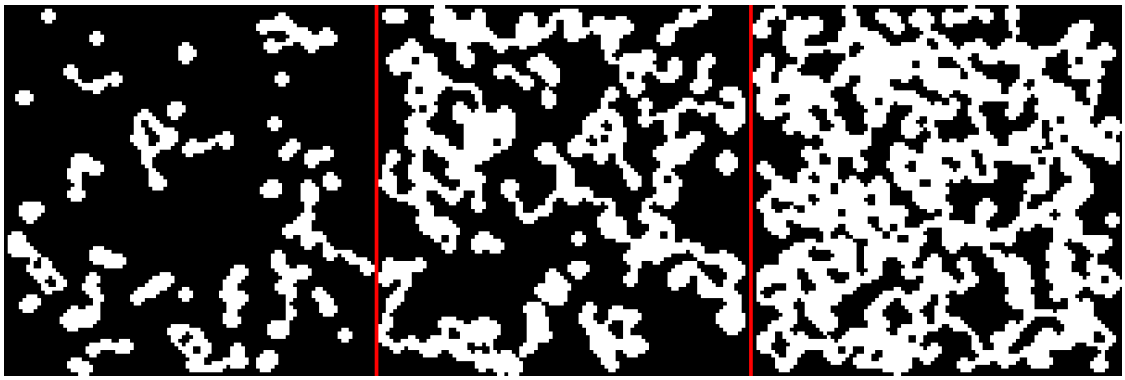


Figura 2. Mapas gerados com diferente proporção de paredes para chão iniciais.

2.2.2. Algoritmo de autômato celular

Para o algoritmo de autômatos cada célula analisa sua vizinhança de Moore contando o número de paredes e aplicando as seguintes regras:

- Se a célula é parede e tem 2 ou menos paredes vizinhas se torna chão.
- Se a célula é chão e tem 5 ou mais paredes vizinhas se torna parede.
- Se a célula não se encaixa em nenhuma das regras acima o seu estado é mantido para a próxima geração.

Se uma célula estiver localizada em um lado, é necessário tratar as células vizinhas que não existem. Por exemplo, uma célula localizada no topo não tem 3 células acima dela. O tratamento dessas células pode contá-las como chão, gerando um mapa com os lados predominantemente sendo chão, ou contá-las como paredes tendo o efeito contrário. Também pode ser feito o tratamento de transformar todas as células de lados em chão ou parede independentemente dos seus vizinhos.

Como em jogos geralmente os lados delimitam o fim do conteúdo de um nível, é indesejável que o jogador saia dessa área. Por isso pode ser feita a escolha de transformar todas as células nos lados como paredes independentemente de suas células vizinhas. Na Figura 3, é mostrado o mesmo mapa mudando o tratamento de seus lados, o primeiro considerando as células inexistentes como chão e o outro sempre forçando os lados a serem paredes.

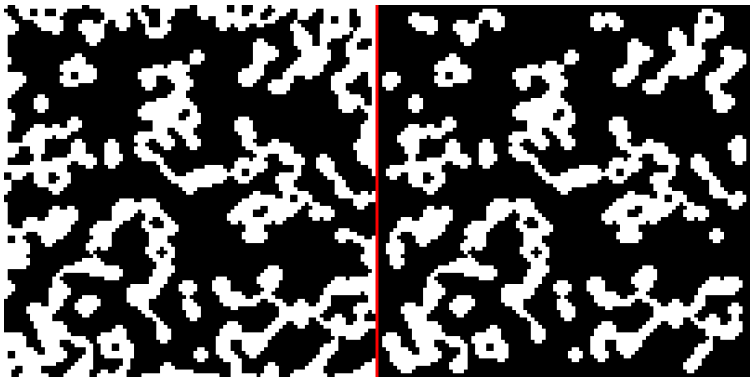


Figura 3. O mesmo mapa com tratamentos de lados diferentes.

2.2.3. O número de gerações

Com as regras utilizadas, cada geração passada tem um efeito gerar grupos maiores de células com o mesmo estado, e aumentar o número total de paredes até um ponto que gerações não geram nenhuma mudança no mapa, momento em que o algoritmo finaliza. Além disso, pode ser utilizada uma regra separada para parar as gerações como sempre executar um número fixo de gerações, ou uma condição separada como alcançar um número especifico de salas, ou uma certa proporção de paredes para chãos. Na Figura 4, é mostrado o mesmo mapa após 3 gerações, 6 gerações, e até que não existam mudanças.

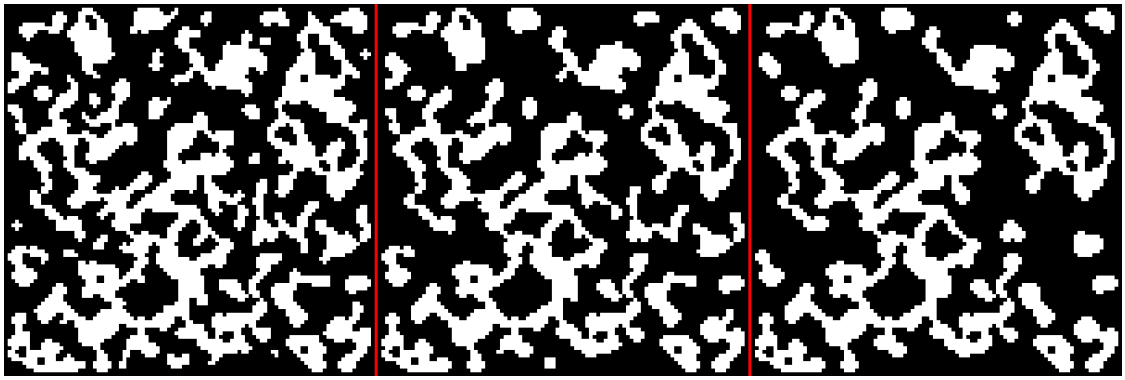


Figura 4. O mesmo mapa com números diferentes de gerações.

2.2.4. Separando em salas

Percorre-se todo o mapa; se uma célula de chão não pertencer a nenhuma sala, é criada uma nova sala com ela sendo adicionada. Verificam-se os vizinhos nas 4 direções ao redor da célula. Todos os chãos vizinhos são adicionados na sala. Esses vizinhos adicionados checam seus vizinhos. Isso se repete até que todas as células ao redor da sala sejam paredes.

3. Trabalhos relacionados

Nesta seção, serão analisados trabalhos envolvidos com a geração de mapas utilizando autômatos celulares, explorando suas metodologias, aplicações e resultados. A análise dos trabalhos correlatos estabelece uma base teórica para a nova pesquisa e possíveis lacunas a serem exploradas.

[Johnson et al. 2010] Analisa os passos necessários para produzir um gerador de mapas com autômatos celulares, É criado um gerador de mapas que gera em matrizes de 50 em 50, o gerador sendo capaz de gerar mais matrizes de 50x50 conectados no bloco inicial, é implementado em um jogo carregando mais blocos para o mapa ao jogador alcançar o fim do mapa permitindo gerar mapas infinitos. Depois é feito uma análise da complexidade e performance do código gerado.

[Fonseca 2016] Faz um estudo de técnicas de geração procedural para cavernas, é feito um gerador utilizando autômatos celulares, e em cima deste mapa gerado é transformado as células em voxels, e utilizando o algoritmo de quadrados marchantes para modificar a renderização das paredes baseado na posição de paredes vizinhas, gerando um mapa final mais curvado que é renderizado dentro do motor Unreal.

[Ashlock and Kreitzer 2020] Utiliza algoritmos genéticos para gerar regras para aprimoramento de mapas com autômatos celulares. Para isso, são geradas matrizes de regras pareadas que juntos geram um mapa, esses mapas são avaliados pelo tamanho de rotas para se conectar em pontos na esquerda direita, acima e abaixo do mapa. Quanto maior o tamanho maior a qualidade, se algum ponto é inalcançável a qualidade é 0. Depois é feita a análise de múltiplos experimentos utilizando parâmetros diferentes de qualidade.

[da Costa and de Oliveira Knop 2021] Implementam um gerador de mapa, separam conjuntos de células passados como salas e criam teleportes entre salas, e criam um

jogo para teste onde o jogador deve coletar moedas e chegar no final de um nível antes que um temporizador expire.

[Minini 2021] Implementa múltiplos algoritmos de geração procedural e demonstra formas de combiná-los. Gera mapas com características de múltiplos algoritmos separados. Utiliza autômatos celulares para gerar partes de mapas mais caóticos e, como pós processamento, modifica mapas gerados utilizando outros algoritmos. Como exemplo para alterar corpos de água, usa AC para separá-los em águas rasas e profundas.

A análise dos trabalhos relacionados demonstra uma diversidade de abordagens e técnicas explorando diferentes aspectos da geração de mapas, os resultados podem ser visualizados na Tabela 1. Enquanto [Johnson et al. 2010] e [Fonseca 2016] se concentram em aspectos fundamentais da geração e renderização de mapas, [Ashlock and Kreitzer 2020] introduzem uma camada de otimização através de algoritmos genéticos. Por sua vez, os trabalhos de [da Costa and de Oliveira Knop 2021] e [Minini 2021] exploram a interatividade e a complexidade dos mapas, incorporando tratamentos separados em cima de mapas gerados por autômatos celulares.

Tabela 1. Tabela comparativa de trabalhos relacionados.

	Johnson 2010	Fonseca 2016	Ashlock 2020	Costa 2021	Minini 2021
Separa mapas em salas	sim	não	não	sim	sim
Cria conexões entre áreas transitáveis	sim	sim	não	não	sim
Popula o mapa com conteúdo	sim	não	não	sim	sim
Implementa dentro de um jogo	sim	não	não	sim	sim
Implementa algoritmos de evolução	não	não	sim	não	não

Apesar das contribuições significativas, ainda existem lacunas que podem ser exploradas. Processos para aprimorar os mapas gerados, a população de mapas com conteúdo, o efeito de parâmetros separados em mapas finais, e a interatividade entre jogadores e mapas são pontos a serem aprofundados.

4. Implementação de algoritmos de geração de mapas com autômatos celulares

Esse algoritmo pode ser separado em algoritmos menores que recebem uma matriz e retornam ela após algum tratamento de: População da *noise grid*, autômato celular, e separação em salas. Nesta seção, serão descritos cada um deles.

4.1. Gerando a matriz inicial

Esse algoritmo é utilizado para gerar a matriz inicial aleatória. Os parâmetros são três inteiros que determinam o número de linhas e colunas da matriz. E a porcentagem de uma célula ser iniciada como parede ou chão. A matriz resultada pode ser utilizada como *seed*, possibilitando o desenvolvedor a gerar o mesmo resultado em sessões diferentes. Este algoritmo pode ser consultado no Algoritmo 1.

4.2. Algoritmo de autômato celular

Esse algoritmo é o que acontece quanto é feito uma geração. O algoritmo recebe uma matriz contendo o estado atual de um mapa, e retorna uma matriz nova contendo os resultados de cada célula aplicado as regras de autômato. Este algoritmo pode ser consultado no Algoritmo 2.

4.3. Separando em salas

O algoritmo percorre por toda a matriz gerando uma sala para cada grupo de chãos vizinhos. a matriz de salas é utilizado junto com a matriz para processamentos futuros. Este algoritmo pode ser consultado no Algoritmo 3.

5. Avaliação experimental

Nessa seção, foram feitos testes separados para cada parâmetro disponível na implementação. Múltiplos mapas foram gerados para cada variação de parâmetro. Por fim, foram analisados os resultados demonstrando os efeitos que uma mudança de parâmetro tem nos mapas finais. Todos os experimentos foram conduzidos com os parâmetros:

- Matrizes de tamanho 50x50.
- Passam por gerações até o mapa não sofrer nenhuma mudança em uma geração.
- 45% de chance de uma célula ser parede na matriz inicial.
- 2 ou menos paredes vizinhas para uma parede se tornar chão.
- 5 ou mais paredes vizinhas para uma chão se tornar parede.

Os algoritmos descritos nas Seções 2 e 4 foram implementados na linguagem TypeScript. O código encontra-se disponível no GitHub[1].

5.1. Resultados

Diferentes tipos de análises foram realizadas. Tais análises estão descritas a seguir.

[1]https://github.com/ifpr-paranavai/MapMaker

Algoritmo 1 Algoritmo de *Noise Grid*.

```
1:  function MAKENOISEGRID(height, width, chanceOfWall)
2:      grid ← empty 2D array
3:      for y ← 0 to height do
4:          grid[y] ← empty array
5:          for x ← 0 to width do
6:              if random number > chanceOfWall then
7:                  grid[y][x] ← 1
8:              else
9:                  grid[y][x] ← 0
10:             end if
11:         end for
12:     end for
13:     return grid
14: end function
```

Algoritmo 2 Algoritmo de autômato celular.

```
1:  function RUNCELLULARAUTOMATA(grid)
2:      newGrid ← empty 2D array
3:      for y ← 0 to length of grid do
4:          newGrid[y] ← empty array
5:          for x ← 0 to length of grid[0] do
6:              count ← COUNTMOORENEIGHBORS(grid, x, y)
7:              if grid[y][x] = 1 then
8:                  if count < 3 then
9:                      newGrid[y][x] ← 0                ▷ Wall becomes floor
10:                 else
11:                     newGrid[y][x] ← 1
12:                 end if
13:             else
14:                 if count > 4 then
15:                     newGrid[y][x] ← 1                ▷ Floor becomes wall
16:                 else
17:                     newGrid[y][x] ← 0
18:                 end if
19:             end if
20:         end for
21:     end for
22:     return newGrid
23: end function
```

5.1.1. Proporção de paredes na grid inicial

Foi gerado um grupo de mapas para cada porcentagem de 0 a 100 totalizando 101 grupos; dados dos resultados podem ser vistos nas Figuras 5. É notável que por todos os cantos serem paredes independente de seus vizinhos, 200 paredes sempre existirão mesmo em um mapa com a matriz inicial contendo apenas chãos.

As porcentagens de 0-27% só contêm mapas com uma sala contendo a maioria da grade. As porcentagens de 78% em diante tem o problema contrario, com os mapas finais compostos apenas por paredes. Isso pode ser ser visto na figura 5(a). Entre esses dois extremos pode ser visto um padrão de que conforme aumenta a porcentagem de paredes iniciais, o número de salas e o tamanho médio diminuem até 49%, onde o número de salas começa a diminuir.

5.1.2. Número de gerações

Foi gerado um grupo de mapas, passando da 1ª geração até a 30ª; dados dos resultados podem ser vistos nas Figuras 6(a) e 6(b). As primeiras gerações têm o maior número de mudanças, com múltiplas salas de 3 ou menos blocos sendo removidas e agregados de chãos se acumulando em salas grandes. Por volta da 10ª geração, o número de mudanças diminui, mas continuam ocorrendo mais remoções de salas pequenas e o aumento do número total de paredes.

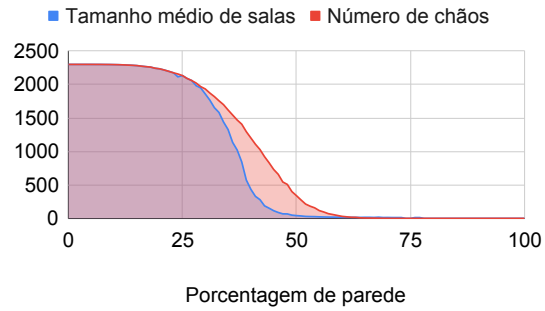
Algoritmo 3 Separando em salas.

```
1:  function GroupCellsIntoRooms(grid)
2:      rooms ← empty array of rooms
3:      for y ← 0 to length of grid do
4:          for x ← 0 to length of grid[0] do
5:              if grid[y][x] = 0 then
6:                  cell ← (x, y)
7:                  cellRoom ← FindCellRoom(rooms, cell)
8:                  if cellRoom is null then                ▷ Cell is floor and not part of a room
9:                      room ← new Room(
10:                     room.img ← GetRandomFloor
11:                     room.cells, room.cornerCells ← empty arrays )
12:                     cellsToAdd ← [cell]
13:                     while cellsToAdd is not empty do
14:                         cellAdd ← cellsToAdd[0]
15:                         floorNeighbors ← FindVonNeumannFloorNeighbors(cellAdd, grid)
16:                         for neighbor in floorNeighbors do
17:                             isNeighborToAdd ← Find(cellsToAdd, neighbor)
18:                             isNeighborAdded ← IsCellInRoom(room, neighbor)
19:                             if isNeighborToAdd = false and isNeighborAdded = false then
20:                                 Push(cellsToAdd, neighbor)
21:                             end if
22:                         end for
23:                         if length of floorNeighbors < 4 then
24:                             Push(room.cornerCells, cellAdd)
25:                         else
26:                             Push(room.cells, cellAdd)
27:                         end if
28:                         Shift(cellsToAdd)
29:                     end while
30:                     Push(rooms, room)
31:                 end if
32:             end if
33:         end for
34:     end for
35:     return rooms
36: end function
```

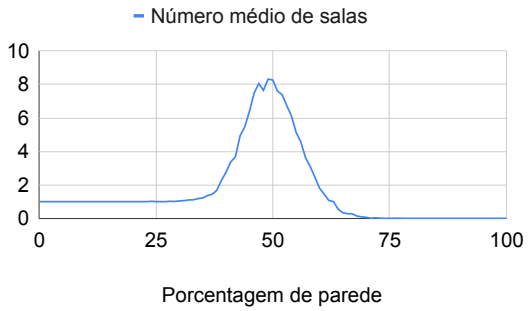
6. Conclusão e trabalhos futuros

Este trabalho analisou o uso de autômatos celulares na geração procedural de mapas para jogos digitais, implementando um gerador e avaliando suas características. Os resultados demonstram que diferentes parâmetros de geração têm impactos significativos nas características dos mapas finais. A análise experimental revelou a flexibilidade do sistema permitindo ajustar parâmetros para obter diferentes tipos de terreno, tornando-o versátil para diversos gêneros de jogos. Mesmo os parâmetros tendo um alcance limitado sem afetar negativamente a qualidade dos mapas finais.

Para trabalhos futuros, sugere-se o desenvolvimento de sistemas para população automática dos mapas com elementos de *gameplay* como inimigos, itens e objetivos. Também pretende-se desenvolver métodos para garantir que os mapas gerados atendam a requisitos específicos de design, como número mínimo de salas ou comprimento de caminho entre pontos-chave.

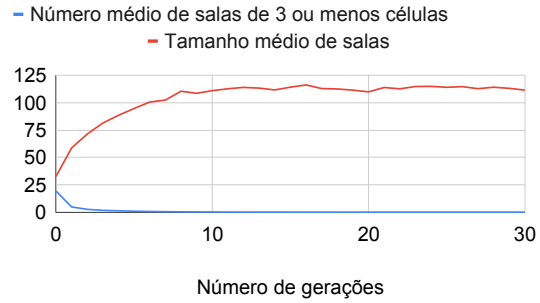


(a) Dados de salas no mapa final por porcentagem paredes na matriz iniciais.

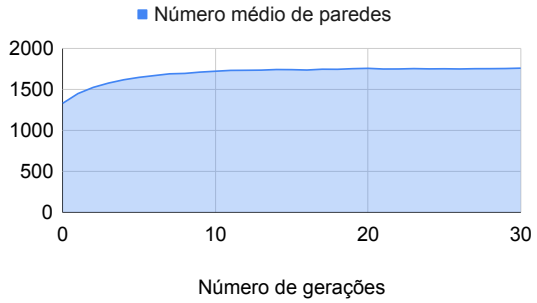


(b) Número médio de paredes no mapa final por porcentagem paredes na matriz iniciais.

Figura 5. Gráficos de características notáveis ao mudar a porcentagem de paredes na matriz inicial.



(a) Dados de salas no mapa final por porcentagem paredes na matriz iniciais.



(b) Número médio de paredes no mapa final por porcentagem paredes na matriz iniciais.

Figura 6. Análise de paredes e salas no mapa final em relação à matriz inicial.

A geração procedural continua sendo uma área promissora para o desenvolvimento de jogos, oferecendo possibilidades de criar conteúdo variado e interessante com custos reduzidos de desenvolvimento. O uso de autômatos celulares demonstrou ser uma abordagem viável e flexível para este fim, merecendo maior exploração e desenvolvimento em trabalhos futuros.

Referências

Ashlock, D. and Kreitzer, M. (2020). Evolving diverse cellular automata based level maps. In Ciancarini, P., Rumpe, B., Gnesi, S., Knapp, A., Hayes, I., and Oliveto, R., editors, *Software Engineering and Formal Methods*, volume 925 of *Advances in Intelligent Systems and Computing*, pages 10–23. Springer Nature Switzerland AG, Guelph, ON, Canada.

Bays, C. (2010). Introduction to cellular automata and conway's game of life. In *Game of Life Cellular Automata*, pages 1–7. Springer.

Castro, M. L. A. and de Oliveira Castro, R. (2008). Autômatos celulares: implementações de von neumann, conway e wolfram. *Revista de Ciências Exatas e Tecnologia*, 3(3):89–106.

da Costa, L. D. and de Oliveira Knop, I. (2021). Autômatos celulares aplicados na geração procedural de conteúdo em jogos. *Revista de Estudos Lúdicos*, 3(3):23–24. Available at Universidade Federal de Juiz de Fora.

Fonseca, C. S. (2016). Geração procedimental de cavernas para jogos digitais 2d na unreal engine. Trabalho de graduação, Universidade Federal de Pernambuco, Centro de Informática, Recife. Avaliador: Giordano Ribeiro Eulalio Cabral.

Johnson, L., Yannakakis, G. N., and Togelius, J. (2010). Cellular automata for real-time generation of infinite cave levels. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*.

Minini, P. P. (2021). Exploração de técnicas atuais de programação de jogos e geração procedural de ambientes através do desenvolvimento de um protótipo de jogo em rust. Trabalho de conclusão de curso, Universidade Federal de Santa Maria, Santa Maria, RS. Defesa realizada por videoconferência.

Oliveira, M. C. d. et al. (2024). Estendendo wavefunctioncollapse com grafos rotulados para a produção de conteúdo para jogos. Trabalho de graduação, Universidade Federal de Uberlândia.

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