Geração Procedural de Mapas para Jogos 2D

Gabriel de Oliveira Barbosa Leite [1] Edirlei Soares de Lima [2]

[1] PUC-Rio, Departamento de Informática, Brasil
[2] UERJ/IPRJ, Departamento de Modelagem Computacional, Brasil

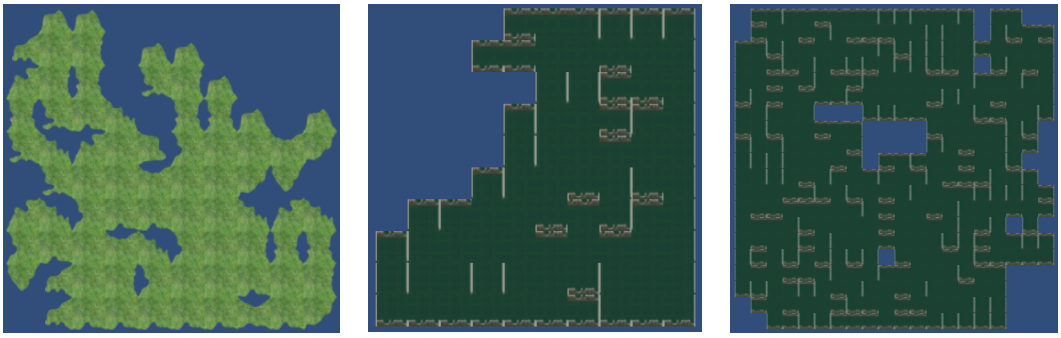

Figura 1: Resultados produzidos pelo algoritmo de geração procedural de mapas proposto.

Resumo

Nos últimos anos os jogos eletrônicos têm se destacado como uma das principais formas de entretenimento digital. Entre os elementos de um jogo, destacam-se os mapas, os quais podem ser construídos de forma pré-definida, onde os desenvolvedores elaboram detalhadamente como será constituído o mapa, ou através de algoritmos baseados em métodos de geração procedural de conteúdo. Um dos maiores desafios no desenvolvimento de algoritmos para geração de mapas é a dificuldade de se criar cenários que sejam, ao mesmo tempo, atraentes e diversificados, permitindo que o jogador possa explorar um novo ambiente a cada sessão de jogo. Este artigo apresenta um algoritmo simples, porém eficiente, para a geração procedural de mapas de cavernas, calabouços e ilhas para jogos 2D.

Palavras-chave: Geração Procedural de Conteúdo, Mapas, Algoritmos.

Contato dos Autores:
gabriel.o.b.leite@hotmail.com
edirlei@iprj.uerj.br

1. Introdução

Nos últimos anos os jogos eletrônicos tem se destacado como uma das principais formas de entretenimento digital. Entre os elementos que compõem os jogos, destacam-se os mapas, os quais são utilizados para auxiliar na localização espacial dos jogadores no ambiente virtual, informando-os sobre o que existe ao seu redor e os perigos que podem existir. A geração destes mapas pode ser feita de forma pré-definida, onde o desenvolvedor elabora detalhadamente como será constituído o mapa, ou através de algoritmos baseados em métodos de geração procedural de conteúdo.

Embora a geração procedural de conteúdo ainda seja pouca utilizada na indústria de jogos, podemos encontrar diversos jogos comerciais que utilizam tais técnicas na geração de parte do seu conteúdo, dentro os quais podemos citar os jogos da série *Diablo* [Blizzard 1996; Blizzard 2000; Blizzard 2012], *Torchlight* [Runic Games 2009; Runic Games 2012] e *Path of Exile* [Grinding Gear Games 2013]. Estes jogos utilizam métodos de geração procedural para a criação de mapas que não são altamente importantes para o andamento da história, o que reduz o tempo de produção sem afetar a jogabilidade e a narrativa.

Uma das maiores dificuldades da área de geração procedural de mapas é a criação de cenários que sejam, ao mesmo tempo, atraentes e diversificados, permitindo que o jogador possa explorar um novo ambiente a cada nova sessão de jogo. Essa característica provê aos jogadores uma experiência única em cada sessão, o que torna o jogo atrativo por mais tempo e aumenta a sua vida útil.

Este artigo apresenta um algoritmo simples, porém eficiente, para a geração procedural de mapas de cavernas, calabouços e ilhas para jogos 2D. O artigo está organizado da seguinte maneira: na Seção 2 são apresentados trabalhos relacionados à geração procedural de mapas em jogos; na Seção 3 são apresentados detalhes sobre a implementação do algoritmo de geração procedural de mapas proposto; e a Seção 4 finaliza este artigo apresentando as conclusões e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

A geração procedural de conteúdo na criação de mapas é uma área emergente no desenvolvimento de jogos. Geralmente, a geração de mapas através de algoritmos baseia-se no processo de seleção dos elementos que constituem os mapas a partir de uma lista de elementos

pré-definidos, os quais são agregados de forma aleatória ou seguindo uma lógica estabelecida previamente pelo programador. Um exemplo que segue este padrão é o jogo *Diablo 3* [Blizzard 2012], no qual existem salas pré-definidas que são espalhadas de forma aleatória pelo mapa e unidas por corredores.

Na literatura podemos encontrar diversos trabalhos que propõem diferentes abordagens para a geração procedural de mapas. Ashlock et al. [2011] apresentam um algoritmo para a geração de labirintos que levava em consideração o número de becos sem saída e o tamanho do caminho entre a entrada e saída desejado como critérios para o algoritmo de geração procedural. Esta técnica foi posteriormente aprimorada por McGuinness e Ashlock [2011] com o objetivo de expandir os mapas gerados utilizando-os como base na criação de mapas maiores. Entretanto, para o caso da geração de mapas utilizando salas pré-definidas ocorre apenas a repetição de salas, o que pode ser ruim dependendo da finalidade do mapa, pois em algumas situações não é válido a repetição de salas com um mesmo evento ou desafio.

Johnson et al. [2010] propõem um algoritmo baseado em autômatos celulares para a geração de cavernas. Este algoritmo utiliza uma matriz de posições, na qual se define um ponto inicial e, a partir dele, o sistema analisa a operação a ser realizada naquela célula e em suas adjacências. Baseado em passos anteriores, o método irá optar por colocar um caminho ou então uma parede na célula corrente. Caso o conteúdo seja uma parede, então ele não fará nada nas bordas adjacentes, caso seja um caminho, então irá analisar as bordas adjacentes. Essa técnica é muito boa quando se tem um terreno simples (caminhos ou paredes), porém a aleatoriedade aplicada pelo algoritmo tende a gerar mapas caóticos. O que é um bom resultado para uma caverna, porém não tão bom quando se deseja uma ilha.

Adams [2002] e Dormans [2011] propõem a geração procedural de mapas utilizando gramáticas gerativas e grafos. Os algoritmos propostos iniciam o processo de geração a partir de um nó do grafo, que representa uma sala ou área do mapa, e selecionam, dentre uma lista de opções, a forma de criação de novas arestas para este nó. As arestas que estiverem apontando para posições nulas ganharão nós não nulos para preencher estes lugares. Para cada novo nó criado, vão sendo geradas novas arestas até que algum pré-requisito seja atingido, seja ele o número de salas do mapa ou o nó corrente não possuir saídas.

Utilizando outra abordagem, Valtchanov e Brown [2012] desenvolveram um algoritmo para geração de mapas com salas de tamanhos e formas diversas, encaixando e adequando-as através de algoritmos genéticos para a geração de mapas grandes. Uma das limitações desta abordagem é que ela aplica-se somente na geração de mapas baseado em salas pré-definidas, não sendo possível gerar uma ilha ou continente de forma atraente.

Uma revisão mais detalhada dos principais métodos de geração procedural de mapas para jogos é apresentada por Linden et al. [2014]. Este tópico também é explorado por Hendrikx et al. [2011] e Carli et al. [2011], que apresentam uma revisão geral da literatura sobre geração procedural de conteúdo para jogos, incluindo a geração de texturas, sons, personagens, ambientes e ecossistemas.

3. Algoritmo de Geração Procedural de Mapas

O algoritmo para geração de mapas procedurais proposto neste artigo divide-se em três etapas: (1) geração recursiva de terrenos; (2) validação de tamanho; e (3) correção da coesão.

O processo de geração recursiva de terrenos inicia-se com a criação de um novo terreno aleatório no centro da matriz que representa o mapa. Recursivamente, a função de geração de terrenos irá expandir o mapa em quatro direções cardiais (norte, sul, leste e oeste) de acordo com terreno criado. Para cada direção, o algoritmo verifica se o terreno possui uma saída, caso possua, então um novo terreno será criado aleatoriamente naquela direção, tendo como restrição o fato de que este novo terreno deverá possuir no mínimo uma saída de retorno para o terreno que o originou.

A Figura 2 mostra os passos do processo de geração recursiva de terrenos para um mapa de tamanho 5x5. Passo (a): Inicialmente é inserido o terreno A no centro do mapa, este terreno possui saídas ao norte (cima), leste (direita) e oeste (esquerda). Passo (b): Em seguida, gera-se o terreno B ao norte, o qual possui saídas ao sul (obrigatória) e a oeste. Passo (c): Na próxima iteração o terreno C é gerado a oeste para ser conectado ao terreno B, este por sua vez somente possui uma saída a leste obrigatória. Passo (d): Como em C e B não possuem nenhuma porta livre, então o algoritmo volta a terminar a execução da função de preencher as portas vazias de A. Em seguida, gera-se um terreno D para se conectar a leste de A, este terreno possui saídas ao oeste e leste. Passo (e): Neste momento gera-se o terreno E para conectar-se a leste de D, este terreno possui apenas uma saída a oeste obrigatória. Passo (f): Como não existem mais saídas em E e D, a execução volta para o terreno A para preencher a ultima porta livre de A. É então gerado o terreno F para se conectar a A, este possui porta de ligação para leste e sul. Passo (g): Ao sul de F é gerado um novo terreno G que possui apenas saída para norte. Passo (h): Como nenhum terreno possui mais portas para área vazias do mapa, então a execução desta etapa é terminada.

A condição de parada da função recursiva é alcançada quando o terreno gerado possui apenas uma

saída, que será usada para ligar com o terreno anterior, ou quando o terreno de destino já estiver previamente ocupado.

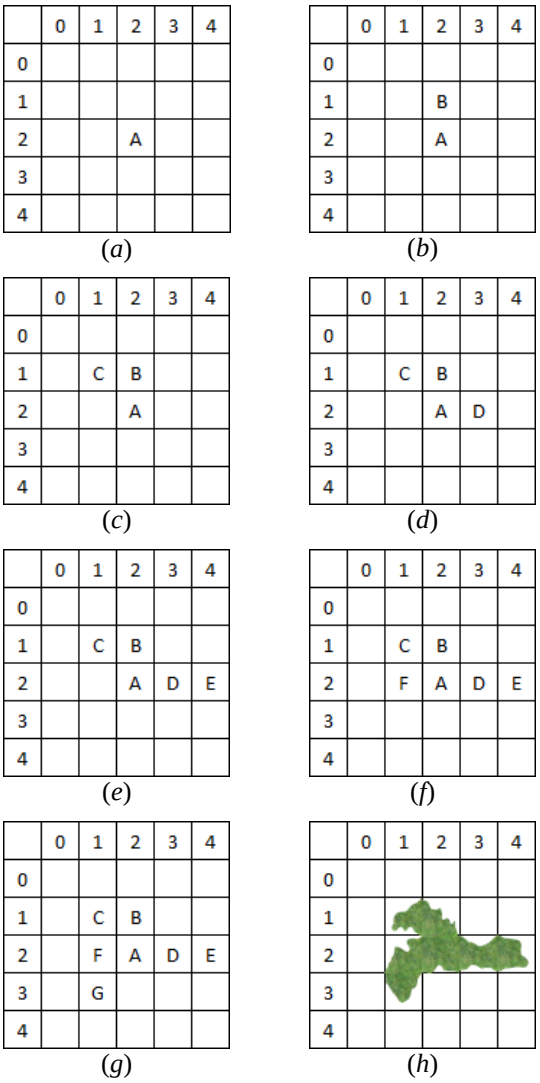

Figura 2: Ilustração dos passos do processo de geração de salas para um mapa de tamanho 5x5.

Embora o processo de geração recursiva de terrenos descrito acima seja capaz de gerar mapas diversificados, a definição de uma ordem fixa para a exploração das quarto direções cardeais tende a gerar mapas que seguem um mesmo padrão de distribuição de regiões. Com o objetivo de evitar esse problema e criar uma distribuição mais uniforme do mapa, foi definida uma estratégia aleatória para a determinação da ordem de criação de novos terrenos, permitindo que todas as direções cardeais tenham chances iguais de serem seguidas. A estratégia consiste em sortear um número de zero a três, caso o número sorteado seja 0, então a chamada recursiva ao processo de geração do terreno será iniciada ao Sul seguida de Norte, Leste e Oeste. Caso o número sorteado seja 1, então a ordem de execução será Leste, Sul, Oeste e Norte. Caso o número seja 2, então a ordem de execução será Norte, Leste, Oeste e Sul. Caso o número seja 3, então a ordem de execução será Oeste, Leste, Norte e Sul.

Após a criação da versão preliminar do mapa pelo processo de geração recursiva de terrenos, o mapa passa por uma etapa de validação de tamanho para garantir que o mapa gerado não ocupe menos que 1/3 da área total especificada. Sempre que um mapa que não atenda a essa especificação for gerado, ele será descartado e a etapa de geração recursiva de terrenos será iniciada novamente. Essa verificação é importante para garantir que mapas muito pequenos não sejam gerados devido às decisões estocásticas tomadas pelo algoritmo.

Após a geração de um mapa que ocupe pelo menos 1/3 da área reservada, o mapa criado passa por um processo de validação e correção para torná-lo coeso. Tornar um mapa coeso significa que uma saída nunca poderá possuir uma parede no terreno adjacente. Para realizar essa tarefa, o algoritmo de correção irá rotacionar cada posição interna da matriz que possuir terrenos com saídas inválidas. Uma saída é considerada inválida quando o terreno de destino daquela saída não possuir uma saída na direção oposta. Conforme o exemplo apresentado na Figura 3, um terreno E na posição [2,1] possui uma saída para Norte, porém o terreno C na posição [1,1] não possui uma saída para sul, então o terreno E é considerado inválido. A cada rotação, se o terreno continuar inválido ele será rotacionado novamente até que o terreno torne-se válido ou até que ele percorra 360º.

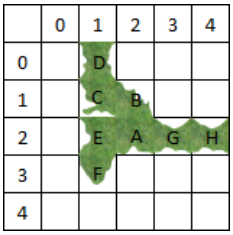

Figura 3: Exemplos de terrenos inválidos (D, E e H).

Após a correção dos terrenos internos, o algoritmo realiza a correção dos terrenos que se encontram nas extremidades do mapa. Caso um terreno da extremidade possua uma saída com ligação para fora da matriz, então a saída será removida. Por exemplo, o terreno D da Figura 3 possui saída para Norte que é considerada invalida, pois não é possível preencher a posição [-1,0] da matriz com um terreno novo.

Caso a rotação individual dos terrenos não tenha sido suficiente para corrigir o mapa, o algoritmo irá buscar pares de terrenos adjacentes que estejam incorretos para rotacioná-los em conjunto. Este processo é realizado através da rotação de um terreno A buscando encontrar uma posição válida para ele em conjunto com outro terreno B. Caso o terreno A percorra 360º sem obter sucesso, então o terreno B será rotacionado 90º e novamente o terreno A será rotacionado até no máximo 360º. Este processo se repete até que A e B se tornem válidos ou até que todas as combinações de rotações possíveis tenham sido testadas.

Caso ainda existam terrenos inválidos no mapa após a execução de todas as correções anteriores, estes passaram por um processo de modificação mais direto de acordo com o seguinte critério: dado um terreno inválido A, ele terá 50% de chance de ter a saída de ligação inválida removida, 25% de chance de criar uma saída para outro terreno adjacente e 25% de não executar nenhuma ação no momento.

Após a conclusão do processo de correção de terrenos inválidos, o mapa produzido pelo algoritmo está pronto para ser utilizado em outras aplicações. Esta integração é realizada através da exportação de um arquivo no formato XML (*eXtensible Markup Language*), contendo toda a estrutura do mapa final gerado pelo sistema. A Figura 1 mostra alguns dos resultados produzidos pelo algoritmo proposto.

4. Conclusões e Trabalhos Futuros

Em geral, os algoritmos de geração procedural de mapas 2D existentes na literatura tratam apenas da criação de salas pré-definidas interconectadas por corredores. O algoritmo proposto neste artigo possibilita a criação de mapas contínuos sem a premissa básica dos outros algoritmos de divisão em salas. Com o algoritmo proposto é possível gerar salas de tamanhos diversos e corredores distintos.

Embora o algoritmo proposto neste trabalho seja capaz de gerar mapas proceduralmente, algumas limitações devem ser consideradas. É importante ressaltar a necessidade da realização de avaliações mais detalhadas do método proposto, levando em consideração as suas capacidades de gerar mapas diversificados e atraentes (do ponto de visto do jogador) e também as suas capacidades de gerar mapas grandes e complexos. Além disso, os mapas gerados pelo algoritmo são compostos por apenas um tipo de terreno, o que é comum em mapas de cavernas e calabouços, mas pode ser inconsistente em mapas de ambientes abertos. Esta ultima limitação pode ser superada com a combinação de vários mapas pequenos gerados pelo algoritmo (cada um de um tipo de terreno diferente), além de ajustes manuais, que podem ser feitos por *level designers*, quando aplicável.

Este trabalho abre diversas possibilidades de trabalhos futuros para o aprimoramento dos resultados iniciais alcançados. Uma destas possibilidades consiste na introdução de um algoritmo genético no processo de geração de mapas, de modo a utilizar o algoritmo proposto para a geração de um conjunto de mapas a serem utilizados como a primeira geração de uma população de mapas do algoritmo genético. A partir desta população e de um critério de avalição (por exemplo, a geração de mapas mais abertos), a próxima geração de indivíduos seria gerada a partir da recombinação dos mapas da geração anterior que possuíssem as melhores avaliações.

Outra possibilidade de expansão do algoritmo proposto neste trabalho consiste em mesclar essa estratégia com as técnicas mais comuns de geração de salas interligadas por corredores, de forma a possibilitar a criação de mapas com algumas salas pré-definidas inseridas em um mapa aberto contínuo.

Referências

ADAMS, D., 2012. Automatic generation of dungeons for computer games. B.Sc. Thesis, University of Sheffield, UK.

ASHLOCK, D., LEE, C., MCGUINNESS, C., 2011. Search-based procedural generation of maze-like levels. IEEE Transactions on Computational Intelligence and AI in Games, vol. 3 (3), 260-273.

BLIZZARD ENTERTAINMENT, 1996. Diablo. CD-ROM.

BLIZZARD ENTERTAINMENT, 2000. Diablo II. CD-ROM.

BLIZZARD ENTERTAINMENT, 2012. Diablo III, DVD-ROM.

CARLI, D.M., BEVILACQUA, F., POZZER, C.T., D'ORNELLAS, M.C., 2011. A survey of procedural content generation techniques suitable to game development. *In: Proceedings of the X Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2011)*, Salvador, Brazil, 26-35.

DORMANS, J., 2011. Level design as model transformation: a strategy for automated content generation. *In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, Article 2.

GRINDING GEAR GAMES, 2013. Path of Exile. Disponível em: https://www.pathofexile.com. Acessado em: 22/07/2015.

HENDRIKX, M., MEIJER, S., VELDEN, J.V.D, IOSUP, A., 2013. Procedural Content Generation for Games: A Survey. ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 9 (1), Article 1.

JOHNSON, L., YANNAKAKIS, G.N., TOGELIUS, J., 2010. Cellular automata for real-time generation of infinite cave levels. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, Article 10.

LINDEN, R., LOPES, R., BIDARRA, R., 2014. Procedural generation of dungeons. IEEE Transactions on Computational Intelligence and AI in Games, vol. 6 (1), 78-89.

MCGUINNESS, C., ASHLOCK, D., 2011. Incorporating required structure into tiles. *In: Proceedings of the 2011 IEEE Conference on Computational Intelligence in Games*, Seoul, 6-23.

RUNIC GAMES, 2009. Torchlight. CD-ROM.

RUNIC GAMES, 2012. Torchlight 2. CD-ROM.

VALTCHANOV, V., BROWN, J. A., 2012. Evolving dungeon crawler levels with relative placement. *In: Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, New York, NY, pp. 27-35.