Análise de métodos de busca para solução de labirintos

Adhonay Junior Silva
Instituto de Ciências Exatas e Informática, PUC-MG
Brasil
adhonay.silva@sga.pucminas.br

Emanuelle Viana Evangelista
Instituto de Ciências Exatas e Informática, PUC-MG
Brasil
emanuelle.evangelista@sga.pucminas.br

Izabela Costa Gonçalves da Silva
Instituto de Ciências Exatas e Informática, PUC-MG
Brasil
icgsilva@sga.pucminas.br

Custodio Junior
Instituto de Ciências Exatas e Informática, PUC-MG
Brasil
custodio@sga.pucminas.br

RESUMO

Este trabalho contém uma análise, baseada no tempo de execução, dos seguintes métodos de busca: Busca em Profundidade, a Busca em Largura, Busca A* e Busca Gulosa. Para isso, foi criado um programa que gera Labirintos e então é possível aplicar sobre estes os métodos de busca citados acima. Foram empregados os métodos de busca em alguns labirintos e então apresentados os resultados na seção Resultados Observados. Ao fim do trabalho é apresentada uma conclusão, em que a partir dessa comparação é definido qual deles apresenta melhor resposta, segundo alguns critérios como tempo de processamento e distância.

PALAVRAS-CHAVE

Grafos, busca, busca em profundidade, busca em largura, busca gulosa, busca A*, labirinto

1 INTRODUÇÃO

Existem diversos problemas que podem ser resolvidos a partir de grafos, e entre os mais comuns observam-se problemas relacionados a caminhamentos. O problema do labirinto representa essa classe de problemas e pode ser encontrado abundantemente em criação de cenários para jogos, por exemplo. Um Labirinto é um conjunto de diversos caminhos que possui uma única entrada e uma única saída, em sua forma tradicional. Cada caminho é feito de forma a tentar confundir quem está nele e dificultar que se chegue na saída.

2 OBJETIVO DO TRABALHO

O objetivo deste trabalho é aplicar métodos de busca a partir da modelagem de grafos para solucionar caminhos de um labirinto e então fazer uma análise dos resultados. Foram aplicados quatro métodos: busca em profundidade, busca em largura, busca gulosa e A*.

3 CONFIGURAÇÕES UTILIZADAS

O trabalho foi desenvolvido na linguagem Java, usando a IDE Netbeans. O problema foi modelado como um grafo. Os pontos em um caminho são os vértices e as arestas o caminho entre eles. A entrada é o vértice origem e a saída o vértice de destino. Assim, a entrada e saída são conectadas de acordo com os caminhos no labirinto. Cada aresta terá um peso associado, que corresponderá à distância para o caminho associado.

4 LABIRINTO

Para gerar o labirinto é usado um algoritmo de geração aleatória de caminhos.

Para gerar os caminhos no labirinto podem ser usados vários algoritmos, como o de Prim ou o de Kruskal. Neste trabalho foi escolhido o "*recursive backtracker*". Este algoritmo, embora necessite de armazenamento de todo o labirinto é rápido e simples de entender, bastando algumas linhas na implementação. Como o trabalho trata de labirintos de tamanho razoavelmente pequeno, não é necessário o uso de outra solução. Esse algoritmo funciona de maneira similar à Busca em Profundidade, mas de forma recursiva, adicionando e removendo paredes de células de acordo com a visitação, enquanto não existirem células não visitadas.

A seguir são apresentados dois labirintos diferentes gerados pelo algoritmo.

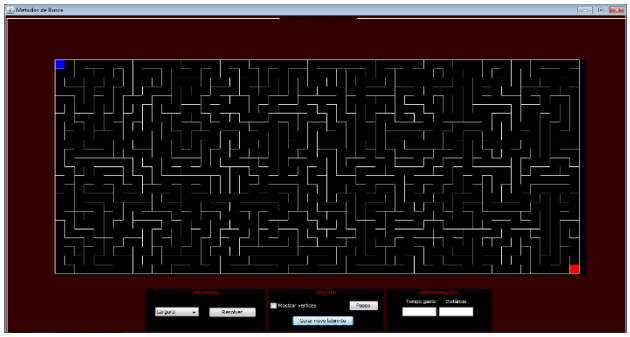

Figure 1: Labirinto 1 gerado pelo algoritmo.

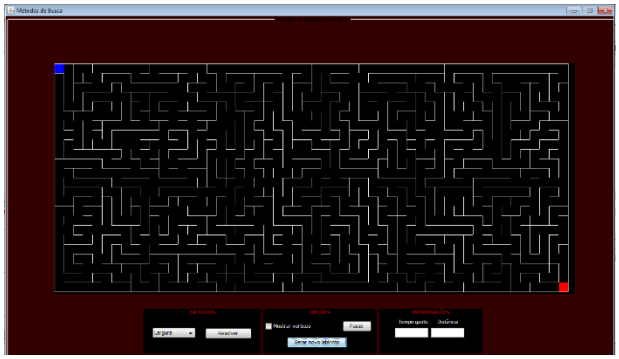

Figure 2: Labirinto 2 gerado pelo algoritmo.

Além do labirinto, a interface do jogo disponibiliza algumas opções. No primeiro bloco podem ser selecionados os métodos de busca que resolveram o labirinto que foi gerado. No segundo bloco de opções é possível gerar um novo labirinto, com novos caminhos e paredes em lugares diferentes. Também se encontra uma opção para que o algoritmo escolhido seja executado passo a passo e visualizar os vértices que representarão o grafo. Por fim, o último bloco apresenta as informações necessárias para as análises desse trabalho, que são a distância e o tempo de execução do método de busca.

5 ALGORITMOS DE BUSCA

Para definir caminhos que levem o jogador da entrada à saída do labirinto, foram implementados quatro algoritmos de busca. A seguir são apresentadas as definições de cada algoritmo usado e seu funcionamento.

5.1 Busca em Profundidade

A busca em profundidade começa no nó raiz e a partir deste aprofunda realizando uma expansão até chegar no nó procurado ou em um nó folha (que não possui filhos). Porém, antes de retroceder explora totalmente cada um dos ramos visitando cada um dos vértices. Para retroceder realiza *backtracking*.

Segue um exemplo de labirinto gerado sendo executado com este algoritmo:

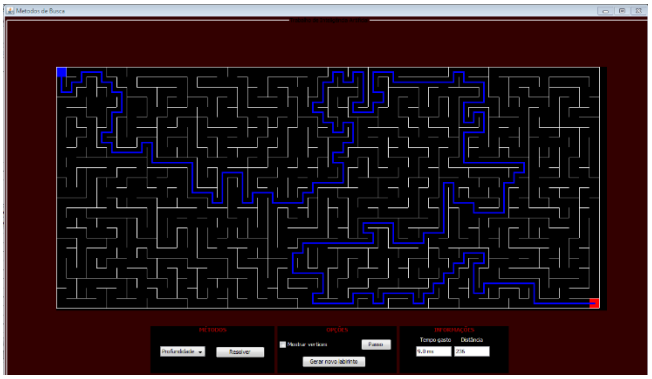

Figure 3: Exemplo de solução do labirinto usando busca em profundidade.

5.2 Busca em Largura

A busca em largura inicia em um determinado nó, que pode ser especificado. A partir deste nó visita os vizinhos mais próximos dele e assim sucessivamente, com o cuidado na escolha dos nós mais próximos a cada nó já visitado.

No trabalho apresentamos uma variação da busca com base em uma árvore. O algoritmo inicia todos os nós indicando que os mesmos foram descobertos. A cada iteração do laço de repetição o algoritmo compara se o nó atual é o final, caso sim ele é retirado da fila. Caso não ele é um nó não visitado, é adicionado à fila e se torna o nó pai. Sendo assim, a sua lista de adjacência se torna a próxima a ser visitada. Após a busca o mesmo se torna o caminho principal do algoritmo de busca até por fim o nó final for o visitado.

Segue um exemplo de labirinto gerado sendo executado com este algoritmo:

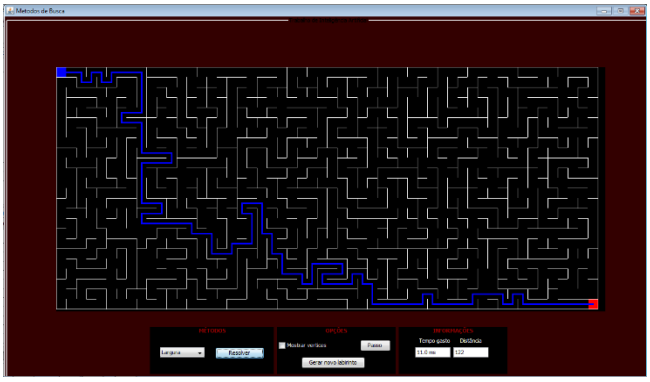

Figure 3: Exemplo de solução do labirinto usando busca em largura.

5.3 Busca A*

Este algoritmo, em cada iteração faz a escolha baseada na combinação g(n) + h(n).Sendo g(n) = uma função de custo que leva em consideração o nó inicial até o nó atual. Sendo h(n) uma função heurística de custo para se chegar do nó atual até o nó desejado.

Segue um exemplo de labirinto gerado sendo executado com este algoritmo:

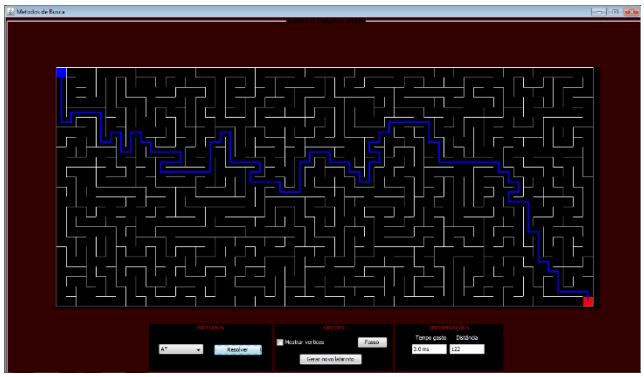

Figure 4: Exemplo de solução do labirinto usando busca A*.

5.4 Busca Gulosa

O algoritmo guloso escolhe, a cada iteração, o próximo caminho corrente mais proveitoso, e o adiciona na solução final. Portanto, faz decisões sem olhar consequências futuras e não volta atrás na decisão (não há *backtracking*). Por essas características, nem sempre encontra o melhor caminho, dentre os possíveis, mas é um algoritmo muito rápido e eficiente.

Para fazer a escolha da alternativa é usada uma função que define como deve ser feita a decisão. Essa função é específica do problema e geralmente atribui valores que determinam a "importância" e capacidade de produzir uma solução da alternativa a ser adicionada, quando for necessário decidir.

A função é chamada heurística e cada vértice tem um valor de heurística associado. A heurística usada neste trabalho foi a Distância Euclidiana, que é a distância entre dois pontos obtida pela fórmula:

$$d(P_1, P_2) \quad = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Segue um exemplo de labirinto gerado sendo executado com este algoritmo:

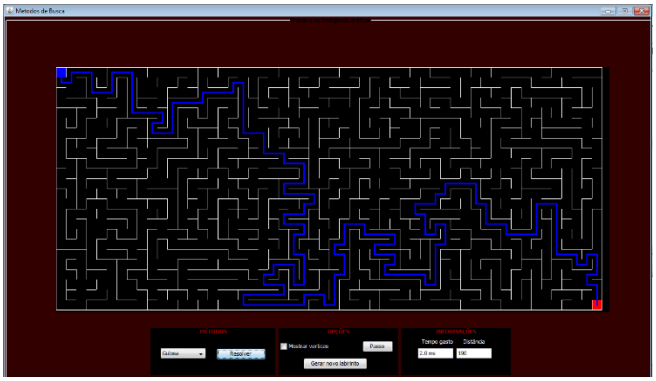

Figure 5: Exemplo de solução do labirinto usando busca gulosa.

Na implementação feita desse algoritmo é usada uma fila que armazena a ordem de visitação dos vértices, adicionando todos os vértices adjacentes ao atual a cada iteração. Dessa forma será feita a decisão de qual deles adicionar na solução. Para cada adjacente é obtido um valor a partir da função heurística e é escolhido o menor valor. Esse processo é feito até que se atinja o vértice de destino.

6 RESULTADOS OBSERVADOS

Executando cada um dos métodos de busca nos labirintos que geramos, conseguimos realizar comparações entre eles. Segue a tabela abaixo com os resultados observados ao executar cada um:

Tabela 1: Desempenho de buscas de profundidade e largura

	Profundidade	Largura
Distância	10	20
Tempo médio (ms)	30	15

Tabela 2: Desempenho de buscas gulosa e A*

	Gulosa	A*
Distância	10	7
Tempo médio (ms)	9	10

6 CONCLUSÕES

Neste trabalho foram usados métodos usuais da teoria de grafos para resolver um labirinto. Através dos resultados obtidos, percebe-se que o algoritmo A* foi o que retornou a solução ótima para o problema, o que era esperado devido às características desse método de busca, já que a heurística usada é admissível, ou seja, para todos vértices o valor da função é sempre menor ou igual ao custo real. Ele retorna então, a menor distância dentre as observadas.

O pior resultado, em tempo de execução foi com a Busca em Profundidade, embora seja um dos mais simples, enquanto a Busca em Largura é mais complexa, mas mais rápida.

Comparando os métodos sem heurística (Profundidade e Largura) e com heurística (Guloso e A*) nota-se que os últimos, em geral, apresentam melhor desempenho em tempo do que os que não usam função para tomar a decisão.

Assim, é possível dizer que a escolha do algoritmo usado deve ser baseada no objetivo da solução do problema e os recursos disponíveis. Enquanto alguns são muito rápidos, outros apresentam sempre uma solução ótima ou são mais ou menos complexos. Dependendo do problema, por exemplo, uma solução quase ótima já é suficiente. Para o problema do labirinto abordado, se a menor distância não importar, pode-se usar o algoritmo Guloso, por exemplo, que mesmo não retornando a solução ótima todas as vezes é mais rápido que as buscas sem heurística e ainda gera um caminho para a saída.

REFERÊNCIAS

[1] Hazim, Nawaf & Al-Dabbagh, Sinan Sameer Mahmood & Naser, Mustafa Abdul Sahib. (2016). Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm. Journal of Computer and Communications. 04. 15-25. 10.4236/jcc.2016.411002.

[2] BUCK, Jamis (Ed.). Maze Generation: Recursive Backtracking. 2010. Disponível em: <http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>. Acesso em: 01 abr. 2018.

[3] LIMA, Edirlei Soares de. Aula 03 – Resolução de Problemas por Meio de Busca. Rio de Janeiro, 2015. 48 slides, color. Disponível em: <http://edirlei.3dgb.com.br/aulas/ia_2015_1/IA_Aula_03_Busca_2015.pdf>. Acesso em: 10 abr. 2018.