

A* para qualquer configuração de vizinhos

Guilherme Caulada¹, Pedro Cacique¹

¹Faculdade de Computação e Informática – Universidade Presbiteriana Mackenzie (UPM)
R. da Consolação, 930 - Consolação, São Paulo - SP, 01302-907 - Brazil

guistoppa1995@gmail.com, phcacique@gmail.com

Abstract. *The A* algorithm is the most used in the game and robotics industry, although it is one of the best pathfinding algorithms out there, the pre-processing necessary in its original implementation makes it hard to execute this algorithm in a dynamic environment. To address that issue, this paper explores the possibility of making A* more flexible and dynamic, so it can adapt to the terrain and other dynamic objects that can be added to the map.*

Resumo. *O algoritmo A* é o algoritmo de busca mais utilizado na indústria de games e robótica, entretanto apesar de ser um dos melhores algoritmos para busca de menor caminho, o pre-processamento necessário em sua versão original dificulta a sua execução em ambientes dinâmicos. Para solucionar este problema este artigo explora a possibilidade de tornar o algoritmo A* mais flexível e dinâmico, para que ele se adapte ao terreno e outros objetos dinâmicos que podem ser inseridos no mapa.*

1. Introdução

A busca do menor caminho é um dos maiores problemas na indústria de jogos e robótica. Entre todos os algoritmos desenvolvidos para solucionar este problema o mais popular de todos é o A*. O algoritmo A* tem sido cada vez mais otimizado, varias modificações e outras versões do algoritmo foram surgindo, buscando sempre aumentar sua velocidade e diminuir seu consumo de memória, sem perder precisão [Cui 2011]. Em busca de tornar o algoritmo mais flexível e adaptável, este artigo sugere uma técnica de otimização que envolve a definição de tipos para as células do mapa que mudam a forma com que o algoritmo encontra seus vizinhos, adaptando o algoritmo as diferentes propriedades do mapa.

2. algoritmos de busca

Existem dois tipos de algoritmos de busca de menor caminho, não-direcionados, como a busca em largura ou em profundidade, estes algoritmos não possuem nenhuma heurística, eles correm cegamente pelo mapa, já os algoritmos direcionados como o Dijkstra e o A* tentam escolher a melhor rota para seu destino. algoritmos direcionados realizam mais iterações do que os não-direcionados, entretanto eles retornam sempre o menor caminho, para qual é dado maior importância [Graham 2003].

2.1. algoritmo A*

O algoritmo A* seleciona uma célula inicial, analisa cada uma das células vizinhas e verifica uma estimativa do custo da rota até a célula destino a partir daquela célula vizinha. Armazenando em cada célula o custo do caminho até ela e a distância estimada dela até a célula destino, o algoritmo A* mistura propriedades de dois algoritmos de busca, Dijkstra e "Melhor-Primeiro". Ele depende de uma função heurística que calcula esta estimativa, quanto mais precisa a função heurística, maior a garantia de que o algoritmo retornará o menor caminho [Stout 1997].

O funcionamento do A* em pseudocódigo [Graham 2003]:

g - custo para chegar até a célula

h - estimativa da distância até o destino

f - soma de g e h, estimativa de qual será o menor caminho

1. Define-se P como célula de início.
2. Define-se os valores f, g e h de P.
3. Adiciona-se P a lista de células abertas.
4. Define-se B como a melhor célula da lista aberta (que possui menor f).
 - a. Se B for a célula destino, o caminho foi encontrado.
 - b. Se não existir células na lista aberta, não existe um caminho.
5. Define-se C como uma célula válida, vizinha a B.
 - a. Define-se os valores f, g e h de C.
 - b. Verifica se C está na lista aberta ou fechada.
 - i. Se estiver na aberta, verifica-se se seu f é menor pelo novo caminho.
 - ii. Se estiver na fechada, atualizamos a sua célula parente.
 - iii. Se não estiver em nenhuma, adiciona-se C a lista de células abertas.
 - c. Repete-se a etapa 5 para todos os vizinhos de B.
6. Repete-se a partir da etapa 4.

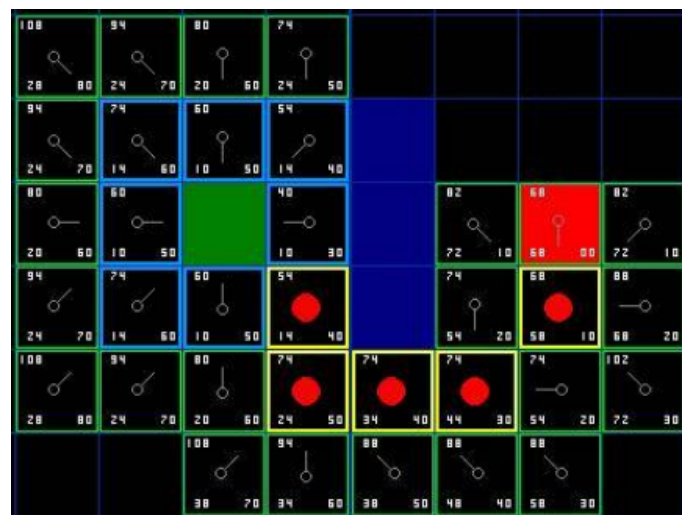


Figure 1. Exemplo do algoritmo A* em funcionamento denotando os valores de f no canto superior esquerdo, g no canto inferior esquerdo, e h no canto inferior direito. [Lester 2005]

2.2. Limitações

Devido ao seu pre-processamento e o alto uso de memória, existe uma grande dificuldade em fazer com que este algoritmo suporte ambientes dinâmicos, uma célula não pode mudar de estado, custo ou quantidade de vizinhos após o pre-processamento, este novo dado será ignorado pelo cálculo de menor caminho do algoritmo. Quanto mais agentes executando o algoritmo, mais recursos de processamento e memória serão necessários para sua execução. Um dos maiores limitadores na indústria de jogos seria o algoritmo A* e o quanto esta indústria depende dele. Em vez de desenvolverem novos conceitos, desenvolvedores preferem modificar suas ideias iniciais para se encaixarem a estas limitações, muitas vezes diminuindo o potencial do produto final [Graham 2003].

2.3. Otimização

Adicionando a cada célula do algoritmo um tipo que informa ao algoritmo como interpretar os vizinhos desta célula específica. Podemos criar células de tipo hexagonais, ou quadradas, ou de qualquer tipo, afinal estas grades de células hexagonais e quadradas com as quais estamos acostumados são apenas representações dos vizinhos de cada célula através de barreiras físicas, nada impede a criação de um tipo de célula que possua uma espécie de túnel que a conecte com uma outra célula que não está conectada fisicamente a ela. Utilizando esta técnica, é possível a criação de um algoritmo A* que se adapte a qualquer tipo de célula que está sendo analisada pelo algoritmo.

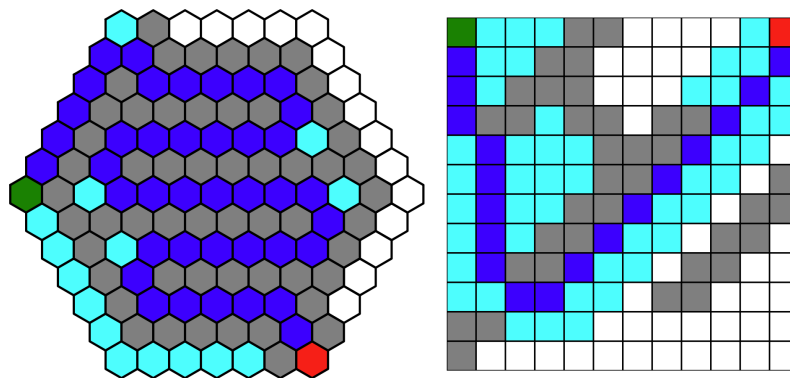


Figure 2. Esta figura demonstra o algoritmo otimizado em execução para grades de células com tipos diferentes.

As grades retangulares, hexagonais, octagonais, de diferentes formatos, são apenas representações de grafos de uma maneira visual, cada uma de suas células podem ser representadas como pontos em um grafo, de maneira que suas conexões representem suas células vizinhas, estas conexões podem ser modificadas de acordo com o tipo da célula.

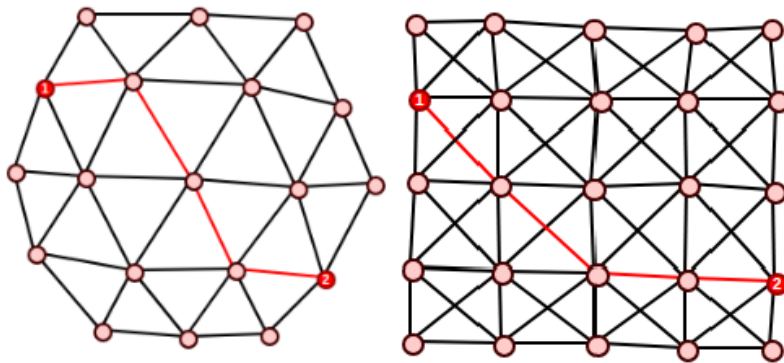


Figure 3. Esta figura representa as grades hexagonais e retangulares no formato de um grafo.

3. Agradecimentos

Agradeço ao professor Pedro Cacique assim como a todos os outros professores da FCI por seus ensinamentos.

References

- Cui, Xiao; Shi, H. (2011). A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130.
- Graham, Ross; Sheridan, S. (2003). Pathfinding in computer games. *The ITB Journal*, 4(2):57–81.
- Lester, P. (2005). A* pathfinding for beginners. pages 1–11.
- Stout, B. (1997). Smart moves: Intelligent pathfinding. *Game Developer Magazine*, pages 1–10.