Dynamic-A: A* para qualquer configuração de vizinhos

Guilherme Caulada[1], **Pedro Cacique**[1]

[1]Faculdade 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 quest for the shortest path is one of the most frustrating problems for the gaming industry. There are several implementations of algorithms that solve this problem, such as Dijkstra's, bread first search, or depth first search, however the A* algorithm is the one that has the best solution for this type of problem. Since its creation this algorithm has received attention from researchers and developers who have created a large list of modifications to the algorithm applying different techniques to improve it. This article describes a modification to the algorithm that seeks to make it more flexible, so that it is able to handle dynamic objects during the calculation of the smallest path. At the beginning you will be presented with an overview of the shortest path search algorithms. Then the A* algorithm will be described in detail as the basis for the optimization presented. Finally, a series of situations where this type of algorithm could be used are presented and we come to a conclusion.*

Resumo. *A busca de menor caminho é um dos problemas mais frustrantes para a industria de jogos. Existem diversas implementações de algoritmos que resolvem esse problema, como o Dijkstra, busca em largura, ou busca em profundidade, entretanto o algoritmo A* é o que possui a melhor solução para este tipo de problema. Desde sua criação esse algoritmo recebeu atenção de pesquisadores e desenvolvedores que criaram uma grande lista de modificações para o algoritmo aplicando diferentes técnicas para aprimora-lo. Esse artigo descreve uma modificação para o algoritmo que busca buscando torna-lo mais flexível, de forma que ele seja capaz a tratar objetos dinâmicos durante o calculo de menor caminho. No inicio sera apresentada uma visão geral sobre algoritmos de busca de menor caminho. Em seguida, o algoritmo A* sera descrito em detalhes como base para a optimização apresentada. Finalmente, uma serie de situações aonde este tipo de algoritmo poderia ser utilizado são apresentadas e chegamos a uma conclusão.*

1. Introdução

A busca do menor caminho geralmente se refere a busca da menor rota entre um ponto inicial, e um ponto final. No nosso dia-a-dia esse tipo de problema aparece em situações mais simples como no transito ou quando vamos fazer compras, e situações mais complicadas, como robôs em uma fabrica ou jogos de computador. Segundo Cui (2011), com o crescimento da industria de jogos, o problema da busca de menor caminho tem se tornado cada vez mais popular e frustrante. Jogos em tempo real geralmente possuem personagens que são enviados de um certo ponto do mapa para um diferente ponto para completar uma certa tarefa. O problema mais comum encontrado na busca de menor caminho em jogos

de computador é como desviar obstáculos e lidar com diferentes tipos de terreno. As primeiras soluções para busca de menor caminho em jogos de computadores, foram logos ultrapassados pelo crescimento da complexidade dos jogos produzidos pela industria.

Devido ao grande sucesso do algoritmo A* muitos desenvolvedores apostam em aumentar sua velocidade para satisfazer as necessidades de seu software. Grandes esforços tem sido feitos nos últimos anos para optimizar esse algoritmo e melhorar sua performance. Exemplos de otimização envolvem novas heurísticas, representações de mapa, estruturas de dados e redução do consumo de memoria.

Neste projeto aproximamos o algoritmo A* de forma diferente, introduzindo uma otimização que busca aumentar a flexibilidade do algoritmo sem grandes impactos em sua performance. O método apresentado, chamado de Dynamic-A, move o calculo de heurística e de vizinhos do A* para cada uma de suas células, dessa maneira cada célula pode mudar seus vizinhos de forma dinâmica, durante o processamento do algoritmo. Cada célula possui em si as informações de como o algoritmo A* deve calcular os seus vizinhos, e esta informação so e acessada enquanto esta célula e analisada, portanto cada célula pode modificar seus vizinhos individualmente, dinamicamente, durante o processamento do menor caminho. Dynamic-A não depende do pre-processamento, não introduz nenhum impacto a performance do algoritmo e sempre encontra o menor caminho.

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 e não consideram o custo do movimento para a célula vizinha, ja 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 ate a célula destino a partir daquela célula vizinha. Armazenando em cada célula o custo do caminho ate ela e a distancia estimada dela ate a célula destino, misturando 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 retornara o menor caminho. [Stout 1997]

O funcionamento do A* em pseudocódigo:

g - custo para chegar ate a célula
h - estimativa da distancia ate o destino
f - soma de g e h, estimativa de qual sera o menor caminho

1. Define-se P como célula de inicio.
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 valida, vizinha a B.
    a. Define-se os valores f, g e h de C.
    b. Verifica se C esta na lista aberta ou fechada.
        i. Se estiver na aberta, verifica-se se seu f é menor pelo novo caminho.
        ii. Se estive 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. [Graham 2003]

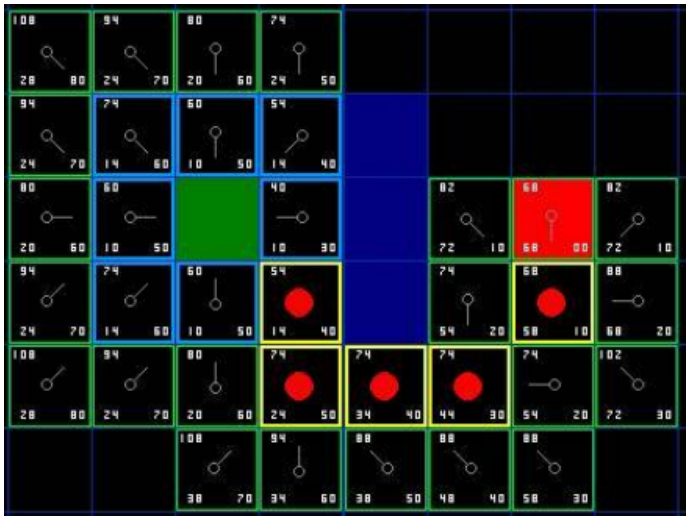

Figure 1. Exemplo do algoritmo A* em funcionamento denotando os valores de f no canto superior esquerdo, g no canto inferior esquerdo, e h no canto inferior direito. [Lester 2005]

2.2. Limitações

Devido ao seu pre-processamento e o alto uso de memoria, 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 apos o pre-processamento, este novo dado sera ignorado pelo calculo de menor caminho do algoritmo. Quanto mais agentes executando o algoritmo, mais recursos de processamento e memoria serão necessários para sua execução. Um dos maiores limitadores na industria de jogos seria o algoritmo A* e o quanto esta industria depende dele. Invés 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].

3. Optimização

Adicionando a cada célula do algoritmo um tipo que informa ao algoritmo como interpretar os vizinhos desta célula especifica. 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 especie de túnel que a conecte com uma outra célula que não esta conectada fisicamente a ela. Utilizando esta técnica, é possível a criação de um algoritmo A* que se adapte a qualquer tipo de célula que esta sendo analisada pelo algoritmo.

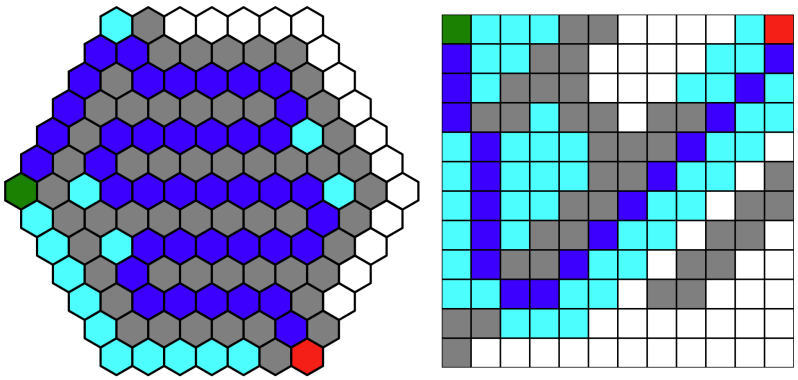

Figure 2. Esta figura demonstra o algoritmo optimizado em execução para grades de células com tipos diferentes.

Ao comparar o algoritmo optimizado com o padrão notamos que a diferença no uso de memoria era minima e so causaria impacto se o numero de células fosse muito alto, devido a possibilidade de células com funções heurísticas diferentes a qualidade destas funções definirão a precisão do resultado final.

3.1. Implementação

As grades retangulares, hexagonais, octogonais, 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 representam suas células vizinhas, estas conexões podem ser modificadas de acordo com o tipo da célula.

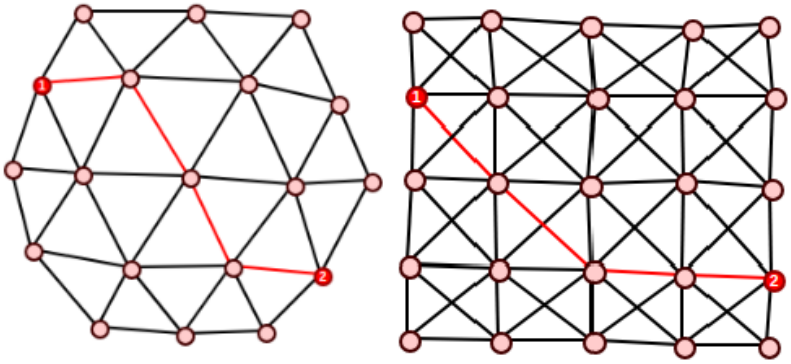

Figure 3. Esta figura representa as grades hexagonais e retangulares no formato de um grafo.

Para implementar a optimização sugerida, utilizamos uma linguagem orientada a objetos e criamos uma classe grafo, que possui elementos da classe célula, cada célula possui funções que dizem sua posição no grafo, seu custo, se é valida ou não, assim como o seu tipo, que define como o grafo calculara os vizinhos da quela célula de acordo com sua posição.

3.2. Utilidade

Este algoritmo poderia ser utilizado em diferentes áreas para a criação de mapas que mudam os dados de suas células dinamicamente. Na área de jogos um mapa dinâmico abre um leque de possibilidades podendo mudar o comportamento da busca de menor caminho dos personagens presentes no jogo. Existem também aplicações no mundo real, no GPS o algoritmo poderia conectar estações de ônibus e oferecer o menor caminho por transporte publico, ou poderia mudar as funções heurísticas do mapa de acordo com o tipo de viagem selecionada mais rápida ou com menos consumo de gasolina.

4. Conclusão

Utilizando está técnica de optimização podemos definir vários tipos diferentes de células, não necessariamente formando grades, seus vizinhos podem ser definidos fora do escopo de suas barreiras físicas abrindo diversas possibilidades para organização de um mapa de maneira que o algoritmo de busca de menor caminho levará em consideração conexões a células que não são suas vizinhas fisicamente assim como suas diferentes heurísticas.

5. 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.