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 [Stout 1997].

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 das posições presentes no mapa, dessa maneira cada posição pode mudar seus vizinhos de forma dinâmica, durante o processamento do algoritmo. Cada posição possui em si as informações de como o algoritmo A* deve calcular os seus vizinhos, e esta informação so e acessada enquanto esta posição é analisada, portanto cada posição pode modificar seus vizinhos individualmente, dinamicamente, durante o processamento do menor caminho. Dynamic-A não depende do preprocessamento, não introduz nenhum impacto a performance do algoritmo e sempre encontra o menor caminho.

2. Algoritmo A*

O algoritmo A* é um algoritmo de busca genérico que pode ser utilizado para diversos problemas, a busca do menor caminho é um deles. Para encontrar o menor caminho o algoritmo A* repetidamente examina o as posições inexploradas que considera mais promissoras. Quando uma posição é explorada o algoritmo para se essa posição é o destino final; caso contrario, ele guarda os vizinhos daquela posição para serem explorados no futuro.

```
1. Adicione a posição inicial a lista de posições em aberto.
2. Repita os seguintes passos:
   a. Procure pela posição que possui o menor f na lista
      de posições em aberto. Defina essa posição como
      a posição atual.
   b. Mova essa posição para lista de posições fechadas.
   c. Para cada posição vizinha a posição atual.
      i. Se estiver na lista de posições fechadas, ignore.
     ii. Se não estiver na lista de posições em aberto
         adicione-a a lista. Defina a posição atual como
         parente dessa posição. Grave o valor de f, g e h
         dessa posição.
    iii. Se ja estiver na lista de posições em aberto,
         cheque se este não é um melhor caminho. Caso seja,
         mude sua posição parente para a posição atual e
         recalcule os valores de f e g.
   d. Pare quando
      i. Adicionar a posição final a lista de posições fechadas.
     ii. Falhar ao buscar a posição final e a lista de posições
         em aberto estiver vazia.
3. A lista de posições utilizadas para se mover da posição
   inicial a posição final é o menor caminho encontrado.
```

Figure 1. Pseudocódigo do A*.

Como padrão quando nos referimos ao A* o valor g de uma posição representa o custo exato do inicio até si, o valor h representa a estimativa de custo ate a posição final e o valor de f é a soma desses dois valores. A Fig. 1 apresenta o algoritmo passo-a-passo.

O A* possui diversas propriedades que foram provadas por Hart, Nilsson e Raphael em 1968. A primeira propriedade é que o algoritmo A* garantidamente trás um caminho entre a posição final e inicial caso ele exista. A segunda, a qualidade do caminho resultante depende em grande parte da heurística escolhida. Para que essa heurística seja admissível ela deve sempre estimar um valor menor ou igual ao menor custo da posição que esta sendo analisada a final. E a terceira propriedade do A* é que ele faz um uso eficiente da função heurística escolhida para ser o método de busca que examina a menor quantidade de posições para encontrar o melhor resultado. Nenhum outro método utiliza uma função heurística para atingir o mesmo objetivo.

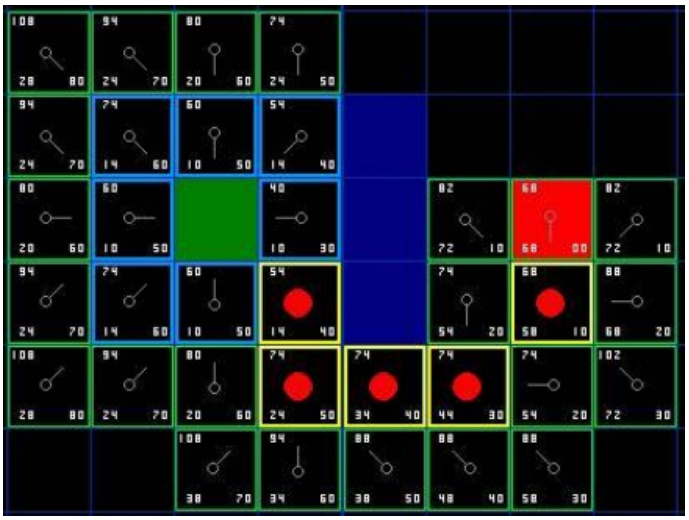

Figure 2. Exemplo do algoritmo A* em funcionamento denotando os valores de f no canto superior esquerdo, g no canto inferior esquerdo, e h no canto inferior direito [Lester 2005].

Apesar do A* ser a escolha mais popular para a busca de menor caminho em jogos de computador, como aplicar esse algoritmo em um jogo depende da sua natureza e representação do mundo virtual. Por exemplo em um jogo que possui uma grade retangular com 1000 posições verticais e 1000 horizontais, com um total de 1 milhão de posições possíveis, sera mais trabalhoso calcular um caminho, reduzindo o espaço de busca pode acelerar significantemente o algoritmo A* [Cui 2011]. A Fig. 2 demonstra um exemplo do A* sendo executado em uma grade quadriculada.

3. Otimizações

As sessões a seguir apresentam otimizações possíveis para o algoritmo A* que foram aplicadas em conjunto com a otimização sugerida neste artigo.

3.1. Espaço de Busca

Em qualquer mapa virtual, os elementos presentes nele devem utilizar uma estrutura de dados para representar esse ambiente de forma que sejam capazes de calcular o caminho

para uma posição destino. Encontrar a melhor estrutura de dados para representar este ambiente é de grande importância para atingir uma performance aceitável para a busca de menor caminho. Como citado no exemplo anterior, um espaço de busca mais simples permite que o algoritmo execute mais rapidamente. A Fig. 3 apresenta exemplos de representações possíveis para um mesmo mapa virtual.

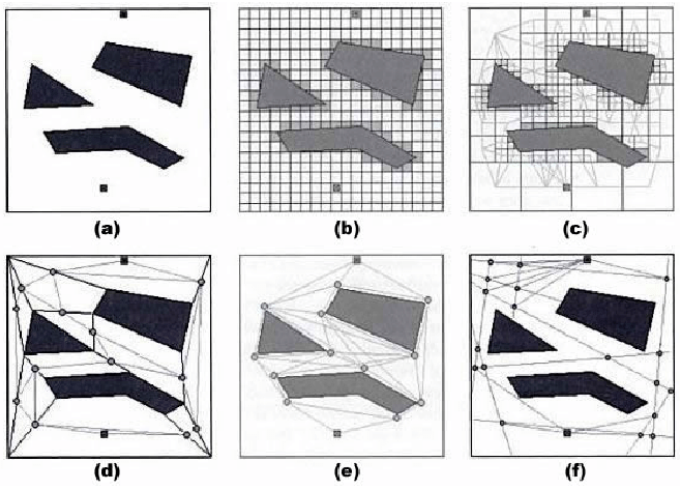

Figure 3. Cinco maneiras de se representar o espaço de busca [Stout 2000].

3.2. Função Heurística

O segredo para o sucesso do A* é que ele estende o algoritmo de Dijkstra introduzindo o uso da função heurística. O algoritmo de Dijkstra garantidamente encontra o menor caminho em um grafo conexo com elementos de diferentes pesos contanto que nenhum de seus pesos possua um valor negativo, entretanto ele não é eficiente pois todos os elementos do grafo devem ser analisados. O algoritmo A* melhora a eficiência computacional significantemente introduzindo o uso de heurística na tomada de decisões. Através da heurística invés de realizar uma busca extensiva em todos os elementos, apenas as posições que aparentam ser boas opções são analisadas. A função heurística utilizada no algoritmo A* estima o custo de uma posição qualquer até a posição destino. Caso o custo estimado seja exatamente igual ao custo real, então apenas as melhores posições são escolhidas e nada mais é analisado. Portanto, uma função heurística que estima o custo com precisão pode tornar o algoritmo muito mais rápido. Por outro lado, utilizar uma heurística que estima um custo um pouco acima do real normalmente resulta em uma busca mais rápida com um caminho aceitável [Steve 2000]. A Fig. 4 apresenta o resultado do algoritmo de busca utilizando diferentes heurísticas para ultrapassar um obstaculo.

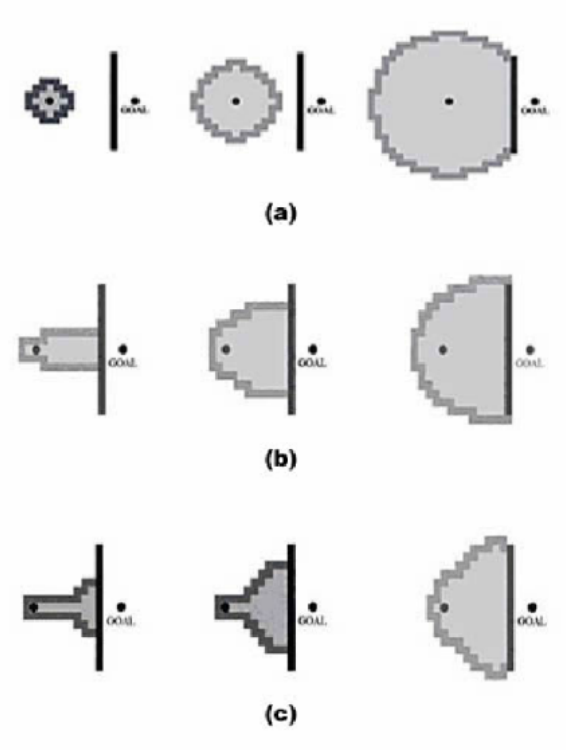

Figure 4. Comparação entre diferentes heurísticas [Steve 2000].

Quando a heurística estima um valor zero (como mostrado na Fig. 4a), o algoritmo A* se torna semelhante ao algoritmo de Dijkstra. Quando a heurística utiliza a distancia euclideana ate o destino (mostrado na Fig. 4b), apenas as posições que aparentam ser melhores opções são examinadas. Quando a heurística estima valores um pouco acima do valor real (como na Fig. 4c), a busca foca nas posições mais próximas do destino final. Apesar de conseguirmos uma execução mais rápida estimando custos acima dos valores reais, quanto acima dos valores reais devem ser estas estimativas é um problema difícil, sem solução ate o momento.

3.3. Estrutura de Dados

Ao ser inicializada uma posição deve ser guardada em algum lugar para rápido acesso. Uma tabela de *hashes* pode ser considerada uma das melhores escolhas pois permite constante acesso aos dados e torna possível descobrir se uma certa posição esta na lista de posições abertas ou fechadas instantaneamente.

Uma fila com prioridades é a melhor maneira para implementarmos e manter as listas de posições, neste caso implementamos uma arvore *heap* binaria. Existem poucos trabalhos introduzindo novas formas de estruturar e manter as listas de posições abertas e fechadas, provavelmente introduzindo uma nova estrutura de dados para o A* pode ajudar a melhorar sua performance significativamente [Cui 2011].

4. Dynamic-A

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.

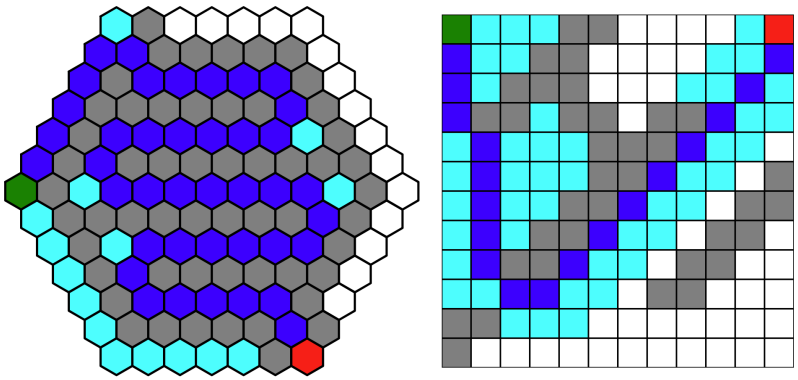

Figure 5. Esta figura demonstra o algoritmo optimizado em execução para grades de células com tipos diferentes.

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.

4.1. 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.

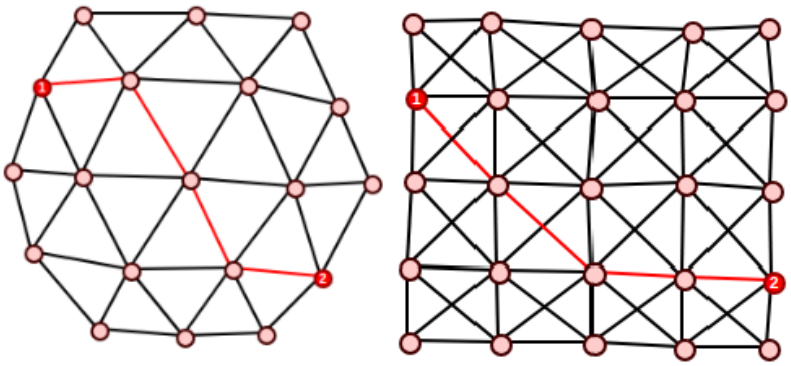

Figure 6. Esta figura representa as grades hexagonais e retangulares no formato de um grafo.

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.

5. Aplicações

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.

6. 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.

7. 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.

Lester, P. (2005). A* pathfinding for beginners. pages 1–11.

Peter, Hart; Betram, R. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans.Syst.Sci.Cybernet*, 4(2):100–107.

Steve, R. (2000). A* speed optimizations. In DeLoura, M., editor, *Game Programming Gems*, pages 264–271. Charles River Media.

Stout, B. (1997). Smart moves: Intelligent pathfinding. *Game Developer Magazine*, pages 1–10.

Stout, B. (2000). The basics of a* for path planning. In DeLoura, M., editor, *Game Programming Gems*, pages 254–263. Charles River Media.