A * Pathfinding para Iniciantes

Por Patrick Lester (Atualizado em 21 de abril de 2004)
(Traduzido por Reynaldo N. Gayoso, rngayoso@msn.com em 04 de Abril de 2005)
(Atualizado em 04 de junho de 2005)

Este artigo foi traduzido em espanhol, portugues, francês, russo, e chinês. Outras traduções são bem-vindas.

O algoritmo A * (pronunciado a-estrela) pode ser complicado para novatos. Existem muitos artigos publicados na internet que explica o A * e a maioria deles é escrito para pessoas que já entendem os fundamentos básicos. Este artigo é para o verdadeiro iniciante.

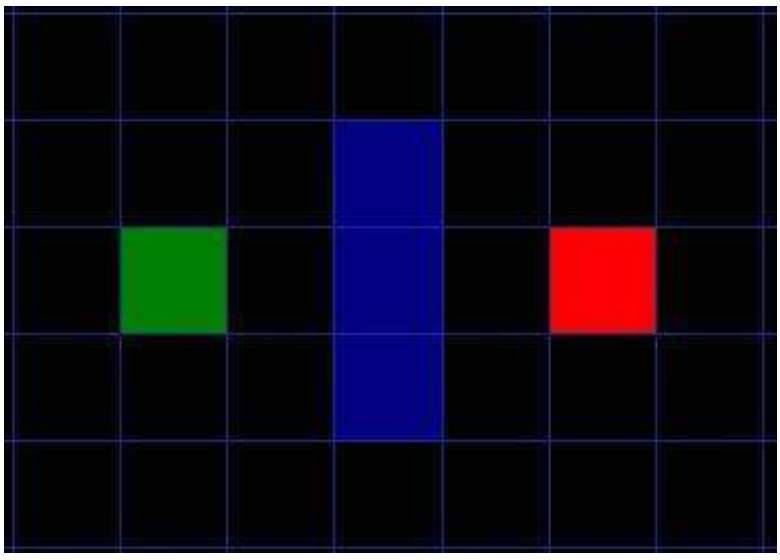
Este artigo também não tenta ser o trabalho definitivo no assunto. Ao invés disto, descreve os fundamentos e prepara você para sair por aí e ler todos esses outros materiais já publicados, tanto na internet quanto em livros, revistas etc., de forma que você possa entender sobre o que eles estão falando. Links para alguns dos melhores artigos são providos ao término deste artigo sob o ítem – Para Ler Mais Adiante.

Finalmente, este artigo não é específico a nenhuma linguagem de programação. Você poderá adaptar o que está escrito aqui a qualquer linguagem de computador. Como você poderia esperar, porém, eu incluí um link a um programa de amostra ao término deste artigo. O pacote de amostra contém duas versões: uma em C++ e outra em Blitz Basic. O pacote também contém os executáveis, se você só quiser ver o A * em ação, como descrito aqui.

Mas, nós estamos pondo a carroça na frente dos bois e indo à frente de nós mesmos. Comecemos pelo princípio...

Introdução: A Área de Procura

Assumindo que nós temos alguém que quer ir do ponto A ao ponto B e que uma parede separa os dois pontos, e isto está ilustrado abaixo, onde o ponto de partida A é verde, o ponto B de destino é vermelho, e os quadrados pintados azuis são a parede entre eles.



[Figura 1]

A primeira coisa que você deveria notar é que nós dividimos nossa área de procura em uma grade quadrada. Simplificando a área de procura, como nós fizemos aqui, é o primeiro passo em pathfinding. Este método particular reduz nossa área de procura a uma ordem simples bi-dimensional. Cada ítem na ordem representa um dos quadrados na grade e seu estado é registrado como passável ou não-passável. O caminho é achado encontrando quais quadrados nós deveríamos tomar para ir de A à B. Uma vez que o caminho é achado, nossa

entidade move do centro de um quadrado ao centro do próximo e assim sucessivamente até que o objetivo é alcançado.

Estes pontos do centro são chamados de nós. Quando você leu em outro lugar qualquer sobre pathfinding, você viu freqüentemente as pessoas discutirem nós. Por que não só chamá-los de quadrados? Porque é possível dividir sua área de pathfinding em algo diferente que não sejam quadrados. Eles poderiam ser retângulos, hexágonos, triângulos, ou qualquer outra forma, realmente. E, os nós poderiam ser colocados em qualquer lugar dentro das formas, no centro ou ao longo das extremidades, ou em qualquer outro lugar. Porém, nós estamos usando este sistema porque é o mais simples.

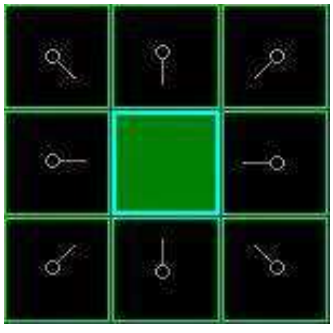
Começando a Procura

Uma vez que nós simplificamos nossa área de procura em um número manejável de nós, como nós fizemos com a grade acima, o próximo passo é administrar uma procura para achar o caminho mais curto. Nós fazemos isto começando do ponto A, conferindo os quadrados adjacentes, e geralmente, procurando para fora até que nós achamos o nosso destino.

Nós começamos a procura fazendo o seguinte:

1. Comece no ponto de partida A e acrescente-o a uma "lista aberta" de quadrados a serem considerados. A lista aberta é como uma lista de compras. Neste momento só há um artigo na lista mas, nós teremos mais depois. A lista contém quadrados que poderiam cair ao longo do caminho você quer tomar, mas, talvez não. Basicamente, esta é uma lista de quadrados que precisam ser verificados.
2. Olhe em todos os quadrados alcançáveis, que podem ser passados por e sejam adjacentes ao ponto de partida, ignorando quadrados com paredes, água, ou outro terreno ilegal. Acrescente-os à lista aberta. Para cada um destes quadrados, salve o ponto A como seu quadrado pai. Este ponto do quadrado pai é muito importante quando nós quisermos localizar nosso caminho. Será explicado mais a fundo posteriormente.
3. Remova o quadrado A de sua lista aberta, e o acrescente a uma lista fechada de quadrados que você não precisa procurar novamente.

Neste momento, você deveria ter algo como a ilustração seguinte. Nesta ilustração, o quadrado verde escuro no centro é seu quadrado pai. É esboçado em azul para indicar que o quadrado foi acrescentado à lista fechada. Todos os quadrados adjacentes estão agora na lista aberta de quadrados a serem conferidos, e eles são esboçados em verde. Cada um dos quadrados tem um ponteiro cinza que aponta para trás, a seu anterior, que é o quadrado pai.


[Figura 2]

Depois disto, nós escolhemos um dos quadrados adjacentes na lista aberta e mais ou menos repetimos o processo anterior, como descrito abaixo. Mas qual quadrado nós escolhemos? O quadrado com o mais baixo custo de F, essa é a resposta.

Graduação do caminho

A chave para determinar quais quadrados usar quando estiver procurando o caminho é a seguinte equação:

F = G + H

onde

- G = é o custo do movimento para se mover do ponto de início até o quadrado determinado na malha seguindo o caminho criado para chegar lá.
- H = é o custo estimado do movimento para mover daquele quadrado determinado até o destino final, ponto B. Isto é frequentemente referido como a heurística, o que pode ser um pouco confuso. A razão pela qual recebe este nome é porque isto é uma adivinhação. Nós não sabemos realmente a distância real entre os pontos até que encontremos o caminho, isto porque podemos ter todos os tipos de coisas no caminho (paredes, água, etc). Você está aprendendo uma maneira para calcular H neste tutorial mas existem muitas outras maneira de se chegar ao mesmo resultado e você poderá encontrá-las em outros artigos existentes na Web.

Nosso caminho é gerado passando repetidamente por nossa lista aberta e escolhendo o quadrado com a mais baixa contagem de F. Este processo será descrito mais adiante em mais detalhe neste artigo. Primeiro, vamos ver mais de perto como nós calculamos a equação.

Como descrito acima, G é o custo do movimento para mover do ponto de partida para o quadrado determinado usando o caminho gerado para chegar lá. Neste exemplo, nós iremos determinar um custo de 10 a cada quadrado horizontal ou vertical movido, e um custo de 14 para um movimento diagonal. Nós usamos estes números porque a distância real para mover diagonalmente é a raiz quadrada de 2 (não se assuste!), ou aproximadamente 1.414 vezes o custo de mover horizontalmente ou verticalmente. Nós usamos 10 e 14 por causa da simplicidade. A relação é quase correta e nós evitamos ter que calcular raízes quadradas e ainda evitamos decimais. Isto porque não é que nós sejamos bobos e não gostemos de matemática. Usando números inteiros, como estes, torna-os muito mais rápidos para o computador. Como você descobrirá logo, pathfinding pode ser muito lento se você não usar atalhos como estes.

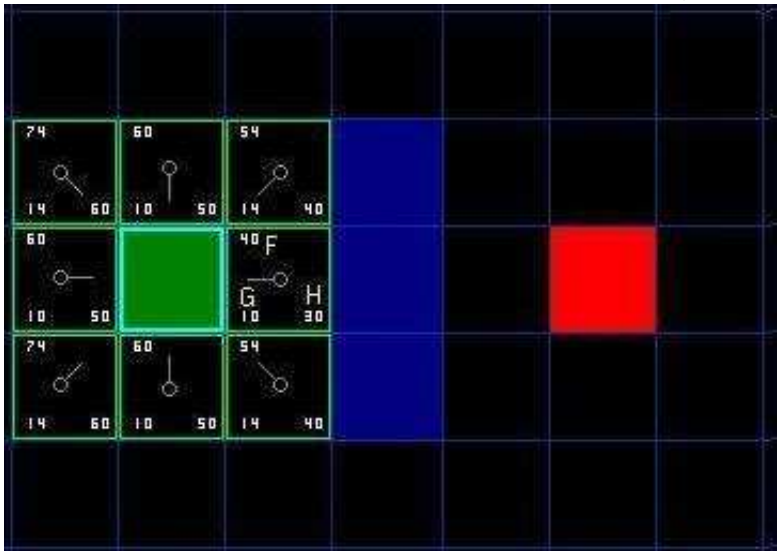
Uma vez que nós estamos calculando o custo de G ao longo de um caminho específico para um determinado quadrado, o modo para entender o custo de G daquele quadrado é tomar o custo de G de seu pai e então somar 10 ou 14 dependendo se é diagonal ou orthogonal (não-diagonal) daquele quadrado pai. A necessidade da utilização deste método ficará aparente um pouco mais adiante neste exemplo, como nós vamos mais de um quadrado de distância do quadrado inicial.

H pode ser calculado de uma variedade de maneiras. O método que nós usamos aqui é chamado o método de Manhattan onde você calcula o número total de quadrados movidos horizontalmente e verticalmente para alcançar o quadrado alvo a partir do quadrado atual, ignorando movimento diagonal, e ignorando qualquer obstáculo que pode estar no caminho. Nós multiplicamos o total então por 10. Isto é, provavelmente, chamado de método de Manhattan porque é como calcular o número de quarteirões da cidade de um lugar para outro, onde você não pode cortar diagonalmente pelo quarteirão.

Lendo esta descrição, você pode adivinhar que a heurística é meramente uma estimativa da distância que falta ser percorrida entre o quadrado atual e o quadrado alvo. Este não é o caso. Nós estamos tentando calcular a distância restante ao longo do caminho de fato (que é normalmente mais distante). O mais próximo que nossa estimativa é à distância restante atual, mais rápido o algoritmo será. Porém, se nós superestimamos esta distância, não é garantido nos dar o caminho mais curto. Nesses casos, nós temos o que é chamado de "heuristica inadmissível".

Técnicamente, neste exemplo, o método de Manhattan é inadmissível porque superestima, ligeiramente, a distância restante. Mas, nós o usaremos de qualquer maneira porque é mais fácil de entender para os nossos propósitos e também porque é só um super-estimação leve. Na rara ocasião quando o caminho resultante não for o mais curto possível, ele será quase tão curto quanto ele pode ser. Quer saber mais? Você pode achar equações e notas adicionais sobre heurística aqui.

F é calculado somando G e H. Os resultados do primeiro passo em nossa procura podem ser vistos na ilustração abaixo. São escritos para o F, G, e contagens de H em cada quadrado. Como é indicado no quadrado imediatamente a direita quadrado inicial, F é impresso em cima na parte esquerda, G é impresso em baixo a esquerda, e H é impresso em baixo a direita.



[Figura 3]

Vamos examinar alguns destes quadrados. No quadrado com as letras, G = 10. Isto é porque ele é só um quadrado do quadrado inicial em uma direção horizontal ou vertical. Os quadrados imediatamente sobre, debaixo de, e à esquerda do quadrado inicial tem o mesmo G = 10. Os quadrados diagonais têm contagens de G = 14.

As contagens de H são calculadas estimando a distância de Manhattan ao quadrado designado vermelho, só movendo horizontalmente e verticalmente e ignorando a parede que está no caminho. Usando este método, o quadrado imediatamente a direita do começo está 3 quadrados do quadrado vermelho, o que totaliza uma contagem de H = 30. O quadrado acima deste quadrado está 4 quadrados do quadrado vermelho (lembre-se, só mova horizontalmente e verticalmente) para uma contagem H = 40. Você agora pode ver como as contagens de H são calculadas para o outro quadrados.

O Valor para F de cada quadrado, novamente, é simplesmente calculado somando G e H.

Continuando a Procura

Para continuar a procura, nós escolhemos simplesmente o quadrado que contém o mais baixo F de todos esses que estão na lista aberta. Nós fazemos então o seguinte com o quadrado selecionado:

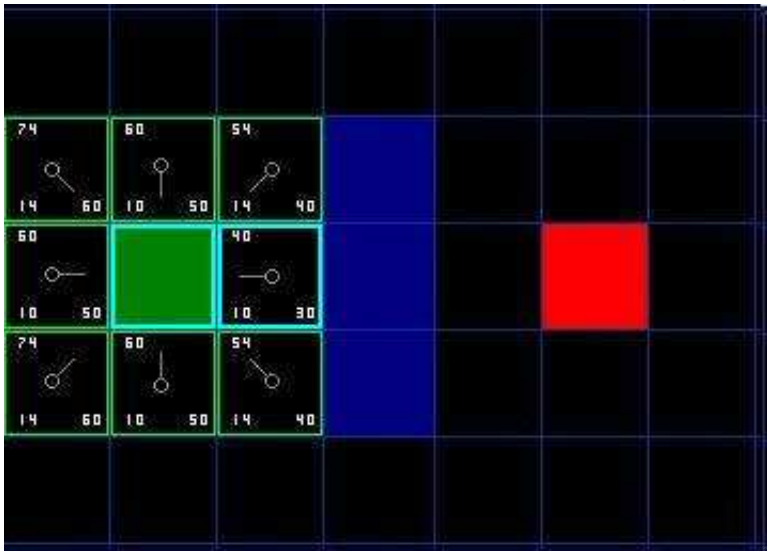
1. Retire-o da lista aberta e o acrescente à lista fechada.

2. Confira tudo dos quadrados adjacentes. Ignorando os que estão na lista fechada ou não-passável (terreno com paredes, água, ou outro terreno ilegal), acrescente quadrados à lista aberta, se eles já não estiverem na lista aberta. Faça o quadrado selecionado o pai dos quadrados novos.

3. Se um quadrado adjacente já estiver na lista aberta, confira para ver se este caminho para aquele quadrado for melhor. Em outras palavras, confira para ver se o G para aquele quadrado é mais baixo se nós usarmos o quadrado atual para chegar lá. Se não for, não faça nada.

4. Por outro lado, se o custo G do novo caminho é menor, troque o pai do quadrado adjacente para o quadrado selecionado (no diagrama acima, mude a direção do ponteiro para apontar para o quadrado selecionado). Finalmente, recalcule o F e G daquele quadrado. Se isto lhe parecer confuso, você verá ilustrado abaixo.

Muito bem, vejamos como isto funciona. Dos nossos 9 quadrados iniciais, nós temos 8 quadrados na lista aberta depois que o quadrado inicial foi posto na lista fechada. Destes quadrados, o quadrado com o mais baixo custo de F é o quadrado imediatamente a direita do quadrado inicial, com uma contagem F = 40. Assim nós selecionamos este quadrado como nosso próximo quadrado. Está destacado em azul na ilustração seguinte.



[Figura 4]

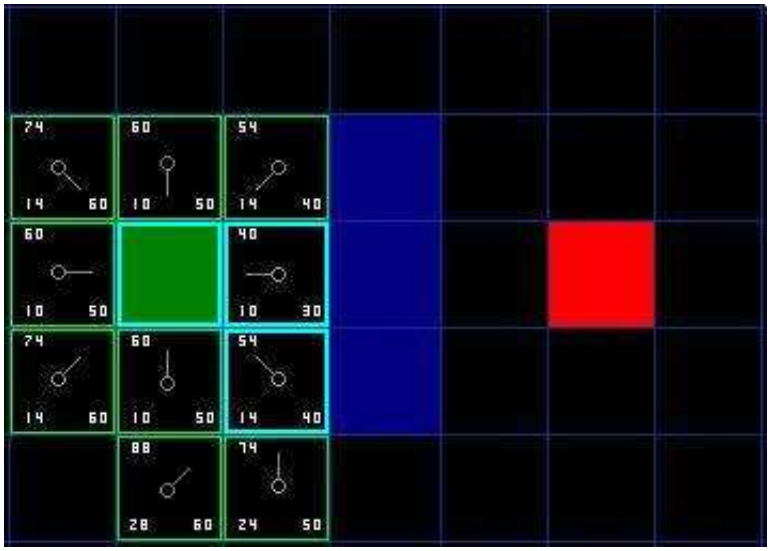
Primeiro, nós o removemos da nossa lista aberta e o acrescentamos a nossa lista fechada (isso é realçado agora em azul). Então, nós conferimos os quadrados adjacentes. Bem, o quadrados imediatamente a direita deste quadrado são paredes, assim nós vamos ignorá-los. O quadrado imediatamente a esquerda é o quadrado inicial e está na lista fechada, assim nós o ignoramos também.

Os outros quatro quadrados já estão na lista aberta, assim nós precisamos conferir se os caminhos para esses quadrados forem melhores usando este quadrado para chegar lá, utilizando as contagens de G como nosso ponto de referência. Olhemos para o quadrado acima do nosso quadrado selecionado. Sua contagem G atual é 14. Se nós passássemos ao invés pelo quadrado atual para chegar lá, a contagem de G seria igual a 20 (10 que é valor G para chegar ao quadrado atual, mais 10 para ir verticalmente). Uma contagem de G = 20 é mais alta que G = 14, assim, este não é um caminho melhor. Isso deve fazer sentido se você olhar para o diagrama. É mais direto para chegar àquele quadrado do quadrado inicial simplesmente movendo diagonalmente, ao invés de mover um quadrado horizontalmente, e então mover um quadrado verticalmente.

Quando nós repetirmos este processo para todos os 4 quadrados adjacentes da lista aberta, nós determinamos que nenhum dos caminhos são melhores passando pelo quadrado atual, assim nós não mudamos nada. Então, agora que nós olhamos todos quadrados adjacentes, nós terminamos com este quadrado, e estamos prontos para mover ao próximo quadrado.

Assim nós seguimos pela lista de quadrados em nossa lista aberta que está agora com 7 quadrados e nós escolhemos um com o mais baixo custo de F. De forma interessante, neste caso, há dois quadrados com uma contagem de 54. Assim qual escolhemos nós? Realmente não importa. Com a finalidade de velocidade, pode ser mais rápido escolher o último que você acrescentou à lista aberta. Isto influencia a procura a favor de quadrados que são achados mais tarde na procura, quando você se chega mais perto do objetivo. Mas realmente não faz diferença. (o tratamento diferenciado de empates é o porque duas versões de A * podem achar caminhos diferentes de comprimento igual.)

Então vamos escolher um logo abaixo e à direita do quadrado inicial, como é mostrado na ilustração seguinte.

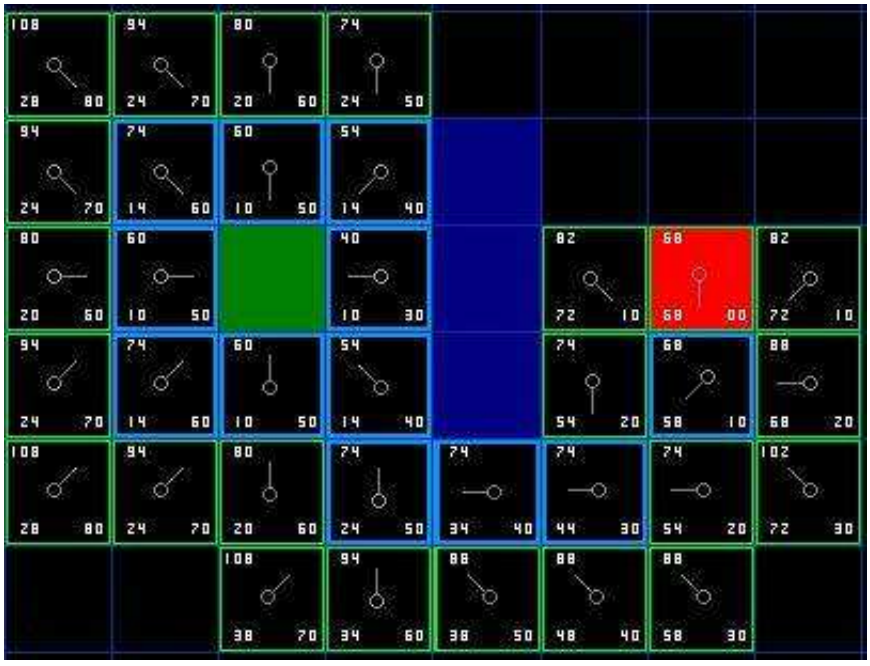


[Figura 5]

Desta vez, quando nós conferimos os quadrados adjacentes nós achamos que o quadrado imediatamente a direita é um quadrado de parede, assim nós o ignoramos. O mesmo acontece para o quadrado que está acima. Nós também ignoramos o quadrado bem debaixo da parede. Por que? Porque você não pode chegar diretamente àquele quadrado vindo do quadrado atual sem cortar pelo canto da parede. Você realmente precisa abaixar primeiro e então mover para cima daquele quadrado, dando a volta no canto neste processo. (Nota: Esta regra em cortar cantos é opcional. Seu uso depende em como seus nós são colocados.)

Isso nos deixa outros cinco quadrados. Os outros dois quadrados debaixo do quadrado atual ainda não estão na lista aberta, assim nós os adicionamos e o quadrado atual se torna o pai deles. Dos outros três quadrados, dois estão já na lista fechada (o quadrado inicial, e o anterior ao quadrado atual, ambos realçados por dentro em azul no diagrama), assim nós os ignoramos. E o último quadrado, imediatamente a esquerda do quadrado atual, é conferido para ver se a contagem de G for mais baixa se você passar pelo quadrado atual para chegar lá. Assim nós terminamos e estamos prontos para inspecionar o próximo quadrado na nosso lista aberta.

Nós repetimos este processo até que nós acrescentamos o quadrado alvo à lista fechada, tal como se parece a ilustração abaixo.

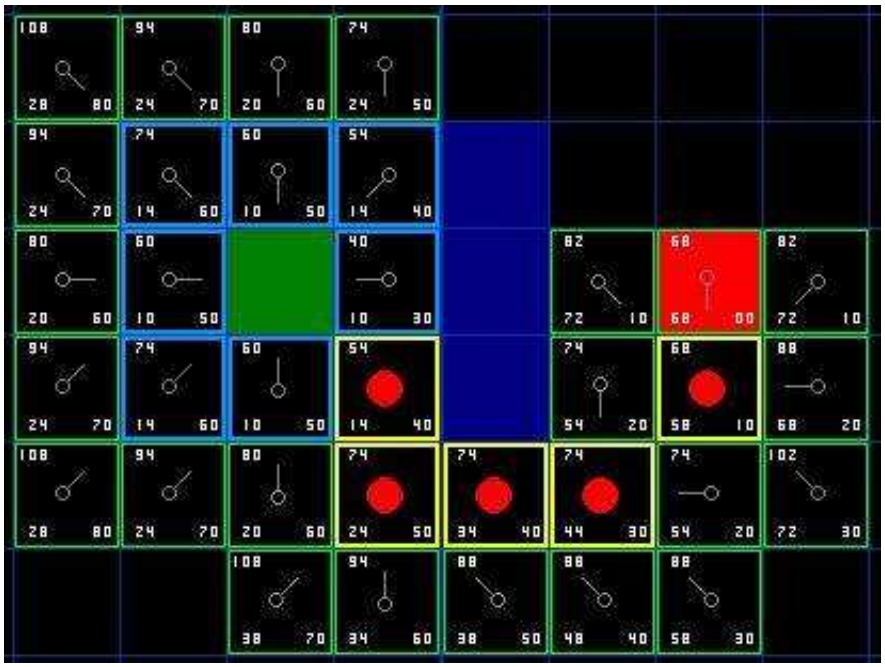


[Figura 6]

Note que o quadrado pai para os dois quadrados abaixo do quadrado inicial mudou da ilustração prévia. Antes ele tinha uma contagem G = 28 e apontou para o quadrado acima e a direita dele. Agora tem uma contagem de 20 e aponta ao quadrado logo acima dele. Isto aconteceu em algum lugar no caminho da nossa procura onde a contagem de G foi conferida e se mostrou ser mais baixa usando um caminho novo. Assim, o pai foi trocado e as contagens de G e F foram recalculadas. Enquanto esta mudança pode não parecer muito importante neste

exemplo, existem muitas situações possíveis onde esta verificação constante fará toda a diferença determinando o melhor caminho a seu objetivo.

Então, como nós determinamos o caminho? Simples, comece do quadrado designado em vermelho, e trabalhe, movendo para trás de um quadrado à seu pai, seguindo as setas. Isto o levará de volta eventualmente ao quadrado inicial, e isto é seu caminho. Deveria se parecer com a ilustração seguinte. Movendo do quadrado inicial A ao quadrado de destino B é simplesmente uma questão de mover do centro de cada quadrado (o nó) para o centro do próximo quadrado no caminho, até que você alcança o objetivo.



[Figura 7]

Resumo do Método A *

Muito bem, agora que você passou pela explicação, vamos descrever o método passo-a-passo todo em um só lugar:

1) adicione o quadrado inicial à lista aberta.

2) Repita o seguinte:

   a. Procure o quadrado que tenha o menor custo de F na lista aberta. Nós referimos a isto como o quadrado corrente.

   b. Mova-o para a lista fechada.

   c. Para cada um dos 8 quadrados adjacente a este quadrado corrente.

      i. Se não é passável ou se estiver na lista fechada, ignore. Caso contrário faça o seguinte:

         1. Se não estiver na lista aberta, acrescente-o à lista aberta. Faça o quadrado atual o pai deste quadrado. Grave os custos F, G, e H do quadrado.

         2. Se já estiver na lista aberta, confere para ver se este caminho para aquele quadrado é melhor, usando custo G como medida. Um valor G mais baixo mostra que este é um caminho melhor. Nesse caso, mude o pai do quadrado para o quadrado atual, e recalcule os valores de G e F do quadrado. Se você está mantendo sua lista aberta ordenada por F, você pode precisar reordenar a lista para corresponder a mudança.

   d. Pare quando você:

i. Acrescente o quadrado alvo à lista fechada o que determina que o caminho foi achado, ou

ii. Não ache o quadrado alvo, e a lista aberta está vazia. Neste caso, não há nenhum caminho.

3) Salve o caminho. Caminhando para trás do quadrado alvo, vá de cada quadrado a seu quadrado pai até que você alcance o quadrado inicial. Isso é seu caminho.

Uma pequena Altercação

Me perdoe por divagar mas vale mostrar que quando você leu várias discussões de A * pathfinding na Web e em foros diversos, você verá alguém ocasionalmente se referir a certo código como A * quando não é. Para o método A * ser usado, você precisa incluir os elementos discutidos acima, especificamente, lista aberta e fechada e caminho usando F, G, e H. Há muitos outros algoritmos de pathfinding, mas esses outros métodos não são A *, o que geralmente é considerado como o melhor das opções disponíveis. Bryan Stout discute muitos deles no artigo referenciado ao término deste artigo, incluindo alguns dos prós e contras. As vezes, alternativas são melhores sob certas circunstâncias, mas você deveria entender onde você está entrando. Muito bem, chega de altercação, de volta ao artigo.

Notas na Implementação

Agora que você entende o método básico, aqui são algumas coisas adicionais a se pensar quando você estiver escrevendo seu próprio programa. Algums dos materiais seguintes referenciam o programa que eu escrevi em C++ e em Blitz Basic, mas os pontos são igualmente válidos em outras linguagens.

1. Outras Unidades (evitar colisão): Se acontecer de você olhar de perto para meu código de exemplo, você notará que ele ignora completamente outras unidades na tela. As unidades passam diretamente através delas. Dependendo do jogo, isto pode ser aceitável ou não. Se você quer considerar outras unidades no algoritmo de pathfinding e quer que se movam ao redor deleas, eu sugiro que você só considere unidades que ou estão paradas ou adjacente na ocasião em que a unidade de pathfinding está calculando o caminho, tratando os locais atuais delas como não-passáveis. Para unidades adjacentes que estão movendo, você pode desencorajar colisões penalizando nós que estejam ao longo dos caminhos respectivos delas, assim encorajando a unidade de pathfinding para achar uma rota alternada (isto é melhor descrito sob ítem 2).

Se você escolheu considerar outras unidades que estão movendo e não estejam adjacentes à unidade de pathfinding, você precisará desenvolver um método para predizer em que ponto onde eles estarão em um tempo qualquer determinado de forma que eles possam ser evitados corretamente. Caso contrário você provavelmente terminará com caminhos estranhos onde unidades farão zig-zag para evitar outras unidades que já não estão mais lá.

Você também vai precisar desenvolver algum código de detecção de colisão porque não importa quão bom o caminho é na ocasião em que é calculado porque coisas podem mudar com o passar do tempo. Quando uma colisão ocorre ou a unidade tem que recalcular um caminho novo ou então, se a outra unidade está movendo e não é uma colisão frontal, espere pela outra unidade sair do caminho antes de proceder com o caminho atual.

Estas idéias são provavelmente bastante que você comece. Se você quiser aprender mais, aqui estão algums links que você poderia achar útil:

- Comportamento direcionado para caracteres autônomos (Steering Behavior for Autonomous Characters): o trabalho de Craig Reynold em dirigir é um pouco diferente de pathfinding, mas pode ser integrado com pathfinding para fazer um movimento mais completo adicionando sistema para evitar colisão.
- O Direcionamento longo e curto em Jogos de computador (The Long and Short of Steering in Computer Games:): Uma pesquisa interessante da literatura em dirigir e pathfinding. Este é um arquivo criado em adobe pdf.
- Movimento de Unidade coordenado (Coordinated Unit Movement): Primeiro de uma série de duas partes de artigos em formação e movimento grupo - baseado no jogo Age of Empires de Dave Pottinger.
- Implementando Movimento Coordenado (Implementing Coordinated Movement): Segundo nas séries de

duas partes de Dave Pottinger.

2. Custo do Terreno Variável: Neste tutorial e no meu programa qua o acompanha, terreno só pode ser duas coisas: passável ou não-passável. Mas, e se você tem terreno que é passável, porém a um custo de movimento mais alto? Pântanos, colinas, degraus em um calabouço, etc.. estes são exemplos de terreno que são passáveis, mas a um custo mais alto do que chão plano, aberto, sem obstáculos. Semelhantemente, uma estrada poderia ter um custo de movimento mais baixo do que o terreno circunvizinho.

Este problema é facilmente resolvido adicionando o custo do terreno quando você estiver calculando o custo de G de qualquer nó determinado. Simplesmente some um custo adicional a tais nós. O algoritmo de pathfinding A * já é escrito para achar o caminho de custo mais baixo e deveria controlar isto facilmente. No exemplo simples que eu descrevi, quando o terreno é só passável ou não-passável, A * procurará o caminho mais curto, mais direto. Mas em um ambiente de custo de terreno variável, o caminho de custo menor poderia envolver viajando uma distância mais longa. Tal como uma estrada ao redor de um pântano em lugar de passar diretamente por ele.

Uma consideração adicional interessante é algo que os profissionais chamam de *influência da cartografia*. Da mesma maneira que com os custos de terreno variáveis descritos acima, você poderia criar um sistema de pontos adicionais e poderia aplicar isto a caminhos para fins de Inteligência Artificial. Imagine que você tem um mapa com um grupo de unidades que defendem uma passagem por uma região montanhosa. Toda vez o computador envia alguém em um caminho por aquela passagem, é golpeado. Se você quisesse, você poderia criar um mapa de influência que penalizam nós onde muita carnificina está acontecendo. Isto ensinaria para o computador favorecer caminhos mais seguros, e ajuda isto a evitar situações bobas onde o computador continua enviando tropas por um caminho particular, só porque é mais curto (mas também mais perigoso).

Ainda, outro uso possível está em penalizando nós que estão ao longo dos caminhos próximos a unidades que estão em movimento. Um dos lados ruins do A * é que quando todo um grupo de unidades tenta achar caminhos a um local semelhante, normalmente há uma quantia significante de sobreposição, como uma ou mais unidades tentam levar as mesmas rotas semelhantes aos destinos delas. Já acrescentando uma penalidade a nós já reivindicados através de outras unidades ajudará assegurar um grau de separação, e redução de colisões. Não trate tais nós como não-passáveis, porém, porque você ainda quer unidades múltiplas para poder pô-las em fila indiana por passagens apertadas, se necessário for. Também, você deveria penalizar só os caminhos de unidades que estão perto da unidade de pathfinding, não todos os caminhos, ou você adquirirá comportamento estranho como unidades que evitam caminhos de unidades que não estão em nenhuma parte, na ocasião, perto delas. Também, você deveria penalizar só nós de caminho que ficam ao longo da porção atual e futura de um caminho, não nós de caminho prévios que já foram visitados e foram deixados para trás.

3. Controlando Áreas Inexploradas: Você já jogou um jogo de PC onde o computador sempre sabe exatamente que caminho tomar, embora o mapa ainda não fosse totalmente explorado? Dependendo do jogo, pathfinding muito bom pode ser irreal. Felizmente, isto é um problema que é e pode ser controlado bastante facilmente.

A resposta é criar um vetor de "Sabe-se Passável" separado para cada dos vários jogadores e oponentes de computador (cada jogador, não cada unidade porque isso requereria muito mais memória de computador). Cada vetor conteria informação sobre as áreas que o jogador explorou, com o resto do mapa assumido para ser passável até provado caso contrário. Usando esta aproximação, unidades vagarão em becos sem saída e farão escolhas erradas semelhantes até que elas aprendam o modo delas ao redor. Porém, uma vez que o mapa é explorado completamente, pathfinding seguiria normalmente.

4. Caminhos mais suaves: Enquanto A * lhe dará automaticamente o caminho mais curto de custo mais baixo, não lhe dará automaticamente o caminho mais suave. Dê uma olhada no caminho final calculado em nosso exemplo (em Figura 7). Naquele caminho, o primeiro passo está abaixo, e à direita do quadrado inicial. Nosso caminho não seria mais suave se o primeiro passo fosse ao invés diretamente o quadrado debaixo do quadrado inicial?

Existem vários modos para resolver este problema. Enquanto você está calculando o caminho você poderia penalizar nós onde há uma mudança de direção, acrescentando uma penalidade às contagens G deles. Alternativamente, você poderia repassar seu caminho depois que é calculado, procurando lugares onde escolhendo um nó adjacente lhe daria um caminho que parece melhor. Para mais informação no assunto, confirme Para um Pathfinding mais Realístico (Toward More Realistic Pathfinding), um artigo de acesso livre mas requer inscrição no site do Gamasutra.com por Marco Pinter.

5. Áreas de Procura Não-quadradas: Em nosso exemplo, nós usamos um plano 2D de quadrado simples. Você

não precisa usar esta aproximação. Você poderia usar áreas irregularmente amoldadas. Pense no jogo de tabuleiro RISK, e os países naquele jogo. Você poderia inventar um enredo de pathfinding para um jogo assim. Para fazer isto, você precisaria criar uma mesa para armazenar quais países são adjacentes a quais, e um custo G associado com mover de um país para o próximo. Você também precisaria propor um método para calcular o H. Tudo o mais seria controlado igual ao exemplo anterior. Em vez de usar quadrados adjacentes, você observaria simplesmente os países adjacentes na mesa ao acrescentar artigos novos a sua lista aberta.

Semelhantemente, você poderia criar um sistema de pontos de direção para caminhos em um mapa de terreno fixo. Geralmente, pontos de direção são atravessados em um caminho, talvez em uma estrada ou túnel em um calabouço. Como o desenhista do jogo, você poderia pre-nomear estes pontos de direção. Dois pontos de direção seriam considerados "adjacentes" um ao outro se não hovesse nenhum obstáculo no caminho diretamente entre eles. Como no exemplo de Risk, você quardaria esta informação de adjacência em uma tabela de para de alguma forma usar-la ao gerar seus artigos de lista abertos novos. Você registraria então o valor G associado (talvez usando a distância de linha direta entre os nós) e valor H (talvez usando uma distância de linha direta do nó para o objetivo). Tudo o mais procederia como sempre.

Amit Patel escreveu um breve artigo que segue adiante além do meu em algumas alternativas. Para outro exemplo de procura em um mapa de RPG isométrico que usa uma área de procura não-quadrada, confirme meu artigo chamado de A * Pathfinding em 2 níveis (Two Tiered A * Pathfinding)

6. Alguns conselhos de Velocidade: Conforme você desenvolve seu próprio programa A * , ou adapta o que eu escrevi, você notará eventualmente que aquele pathfinding está usando um pedaço grande de seu tempo de CPU, particularmente se você tem um número decente de unidades de pathfinding no tabuleiro e um mapa razoavelmente grande. Se você leu os materiais disponíveis na Web, você achará que isto é até mesmo verdade para os profissionais que projetam jogos como Starcraft ou Age of Empires. Se você ver os processos começarem a reduzir a velocidade devido ao uso do pathfinding, aqui são algumas idéias que podem fazer os processos andarem mais depressa:

- Considere um mapa menor ou menos unidades.
- Nunca faça pathfinding para mais de alguns unidades de cada vez. Ao invés disto, ponha-os em uma fila e os tire fora dela em vários ciclos de jogo. Se seu jogo estiver correndo a, digamos, 40 ciclos por segundo, ninguém notará. Mas eles notarão se o jogo parecer reduzir a velocidade de vez em quando exatamente quando todo um grupo de unidades for calculando pathfinding ao mesmo tempo.
- Considere usando quadrados maiores (ou qualquer outra forma você está usando) para seu mapa. Isto reduz o número total de nós procurados para achar o caminho. Se você for ambicioso, você pode inventar dois ou mais sistemas de pathfinding que são usados em situações diferentes, dependendo do comprimento do caminho. Isto é o que os profissionais fazem, usando áreas grandes para caminhos longos, e trocando então para procuras melhores que usam areas menores quando você chega perto do objetivo. Se você estiver interessado neste conceito, confirme meu artigo A * Pathfinding em 2 níveis (Two-Tiered A * Pathfinding).
- Para caminhos longos considere utilizar caminhos pré-calulados que são parte do jogo.
- Considere pré-processar seu mapa para descobrir que áreas são inacessíveis do resto do mapa. Eu chamo estas áreas de ilhas. Em realidade, elas podem ser ilhas ou qualquer outra área que são cercadas e inacessíveis. Um dos lados ruins do A * é que se você disser a ele que procure caminhos a tais áreas, ele procurará o mapa inteiro, só parando quando todos os quadrados/nós acessíveis forem processados pelas listas abertas e fechadas. Isso pode desperdiçar muito tempo de CPU. Pode ser prevenido predeterminando quais áreas são inacessíveis (por flood-fill ou rotina semelhante), registrando aquela informação em um vetor de alguma ordem, e conferindo isto antes de começar uma procura de caminho.
- Em um ambiente cheio de labirintos, considere marcar nós que não conduzem em lugar algum como becos sem saída. Podem ser pré-designadas tais áreas manualmente em seu editor de mapa ou, se você for ambicioso, você poderia desenvolver um algoritmo para identificar tais áreas automaticamente. Qualquer coleção de nós em uma determinada área de beco sem saída poderia ser dada um número identificando sem igual. Então você poderia ignorar todos os becos sem saída seguramente quando estiver executando o pathfinding, só parando para considerar nós em uma área de beco-sem-saída se o local do

começo ou destino acontecerem de estar na área de beco-sem-saída particular em questão.

7. Mantendo a Lista Aberta: Este é de fato o elemento que consome a maioria do tempo do algoritmo A * de pathfinding. Toda vez você acessa a lista aberta, você precisa achar o quadrado que tem o mais baixo valor de F. Há vários modos que você poderia fazer isto. Você poderia salvar os ítens de caminho conforme a necessidade, e simplesmente passar pela lista cada vez que você precisar achar o quadrado com o mais baixo valor de F. Isto é simples, mas realmente reduz a velocidade para caminhos longos. Isto pode ser melhorado mantendo uma lista ordenada e simplesmente pegando toda vez o primeiro artigo da lista toda vez que você precise do quadrado com o valor F mais baixo. Quando eu escrevi meu programa, este foi o primeiro método que eu usei.

Isto funcionará razoavelmente bem para mapas pequenos, mas não é a solução mais rápida. Programadores sérios de A * que querem velocidade real usam algo chamado de 'binary heap', e isto é o que eu uso em meu código. Em minha experiência, esta aproximação será pelo menos 2 a 3 vezes mais rápida na maioria das situações, e geometricamente mais rápido (mais de 10 vezes mais rápida) em caminhos mais longos. Se você é motivado para descobrir mais sobre 'binary heap', confirme meu artigo, Usando heaps binários em A * Pathfinding (Using Binary Heaps in A* Pathfinding).

Outro estrangulamento possível no processamento é o modo pelo qual você limpa e mantém suas estruturas de dados entre as chamadas de pathfinding. Eu, pessoalmente, prefiro armazenar tudo em vetores. Enquanto os nós podem ser gerados, gravados, e podem ser mantidos de uma maneira dinâmica, orientada a objeto, eu acho que a quantidade de tempo necessário para criar e apagar tais objetos soma um tempo adicional que por sua vez reduz a velocidade do processamento. Porém, se você usa vetores, você precisará limpá-los entre chamadas. A última coisa que você quererá fazer em tais casos é passar tempo zerando tudo depois de uma chamada de pathfinding, especialmente se você tem um mapa grande.

Eu evito este custo indireto criando um vetor 2D, se preferir, matrix bi-dimensional, chamado QualLista(x,y) que designa cada nó em meu mapa como estando na lista aberta ou na lista fechada. Depois de tentativas de pathfinding, eu não zero este vetor. Ao invés disto eu reinicio os valores de naLista Aberta e naListaFechada em toda chamada de pathfinding incrementando ambos em +5 ou algo semelhante em cada tentativa de pathfinding. Deste modo, o algoritmo pode ignorar seguramente como lixo qualquer dado deixado de tentativas de pathfinding prévias. Eu também armazeno valores como custo de F, G e H em vetores. Neste caso, eu escrevo simplesmente em cima de qualquer valor anterior e não me aborreço limpando os vetores quando eu termino.

Dados armazenando em vetores múltiplos consomem mais memória mesmo assim há uma escolha. No final das contas, você deveria usar qualquer método você sente que você está muito confortável em trabalhar.

8. O Algoritmo de Dijkstra: Enquanto A * geralmente é considerado como o melhor algoritmo de pathfinding (veja discurso acima), há um outro algoritmo que tem seus usos, pelo menos o algoritmo de Dijkstra. Dijkstra é essencialmente igual a A *, excetuando-se que não há nenhuma heurística (H sempre igual a 0). Porque não tem nenhum heurístico, a procura é feita se expandindo igualmente para fora em todas as direções. Como você poderia imaginar, por causa disto, Dijkstra normalmente acaba explorando uma área muito maior do mapa antes do objetivo ser achado. Isto geralmente o faz ser mais lento que o A *.

Então, por que usá-lo? Às vezes nós não sabemos onde nosso destino alvo está. Imagine que você tem uma unidade junta-recursos que precisa ir adquirir alguns recursos de algum tipo. A unidade pode saber onde várias áreas de recurso estão, mas quer ir para a mais próxima. Aqui, Dijkstra é melhor que A * porque nós não sabemos qual é o mais próximo. Nossa única alternativa é usar repetidamente A * e achar a distância para cada um, e então escolher aquele caminho. Há situações semelhantes provavelmente incontáveis onde nós sabemos o tipo de local que nós poderíamos estar procurando, queira achar a mais próxima, mas não sabemos onde é ou qual poderia ser a mais próxima.

Para Ler Mais Adiante

Muito bem, agora você tem os fundamentos básicos e um senso de alguns dos conceitos avançados. Neste momento, eu sugestionaria que voce olhasse meu código fonte. O pacote contém duas versões, uma em C++ e uma em Blitz Basic. Ambas as versões são comentadas pesadamente e deveriam ser bastante fácil seguir, relativamente falando. Aqui é o link: Programa Exemplo: A * Pathfinder (2D) Versão 1.9 (Sample Code: A* Pathfinder (2D) Version 1.9).

Se você não tiver acesso a C++ ou Blitz Basic , podem ser achados dois arquivos exe pequenos na versão de C++. A versão de Blitz Basic pode ser executada carregando a versão grátis de Blitz Basic 3D (não é Blitz Plus)

no web site do Blitz Basic. Uma demonstração on-line por Ben O'Neill pode ser achada aqui.

Você também deveria considerar uma leitura pelas páginas seguintes na Web. Elas deveriam ser muito mais fáceis entender agora que você leu este tutorial.

- Páginas do Amit sobre o A * (Amit's A* Pages): Esta é uma página amplamente referenciada por Amit Patel, mas pode ser confuso se você não leu este artigo primeiro. Vale a pena verificar. Veja, especialmente, os próprios pensamentos de Amit no tópico em questão.
- Movimentos inteligentes: Pathfinding Inteligente (Smart Moves: Intelligent Path Finding): Este artigo por Bryan Stout no Gamasutra.com exige inscrição pata ler. A inscrição é grátis e vale a pena só para ler este artigo, muito menos os outros recursos que estão lá disponíveis. O programa escrito em Delphi por Bryan me ajudou a aprender A *, e é a inspiração por trás do meu programa A *. Este artigo também descreve algumas alternativas para A *.
- Análise de terreno (Terrain Analysis): Isto é um artigo avançado, mas interessante, por Dave Pottinger, um profissional do Ensemble Studios. Este sujeito coordenou o desenvolvimento de Age of Empires e Age og Kings. Não espere entender tudo aqui, mas é um artigo interessante que poderia lhe dar algumas idéias para o seu próprio. Inclui alguma discussão de mip-mapping, influência de mapa, e alguns outros conceitos de IA/pathfinding avançados.

Alguns outros web sites que valem a pena serem vistos:

- aiGuru: Pathfinding (http://www.aiguru.com/pathfinding.htm)
- Game AI Resource: Pathfinding (http://www.gameai.com/pathfinding.html)
- GameDev.net: Pathfinding (http://www.gamedev.net/reference/list.asp?categoryid=18#94)

Bem, é isto. Se você escrever um programa que usa quaisquer destes conceitos, eu gostaria de vê-lo. Você pode me escrever no seguinte correio eletrônico:

patrick@policyalmanac.org

Até então, boa sorte!