ALGORITMO DE *PATH FINDING* PARA ALVOS MÓVEIS[1]

PATH FINDING ALGORITHM FOR MOBILE TARGETS

Fabricio Santini Einloft[2]
Guilherme Dhein[3]

RESUMO

Neste trabalho, foi desenvolvido um algoritmo para procurar o melhor caminho que entidades (personagens) de jogos devem percorrer em um ambiente dividido em células. Parte destas células são ocupadas por obstáculos, impedindo que o personagem cruze por elas. Outra é a célula objetivo, ocupada pelo alvo que o personagem deve atingir. A partir dela, são calculados valores crescentes para as outras células do ambiente. O melhor caminho será encontrado seguindo os valores das células, em ordem decrescente, até o ponto em que se localiza o alvo. O alvo pode ser móvel, ou seja, a célula objetivo pode mudar a execução do jogo. Para tratar esta dificuldade, foram consideradas áreas que agregam um certo número de células, dentro das quais o alvo pode se mover sem que os valores das células sejam recalculados.

Palavras-chave: *path finding*, algoritmos de menor caminho, jogos de computador.

ABSTRACT

In this work, it was developed a algorithm for finding the best path for entities from games to follow in a cell divided environment. Obstacles occupy some cells, so the entity is not able to cross those. Another one is the goal cell, witch is occupied by the target that must be achieved by the entity. After this cell, all the other cells from the environment become increasing values. The best path is obtained by following the cells values, in a decreasing order, to the point were the target is. The target may be mobile, what means that the goal cell may change during the game execution. To avoid this problem, areas that aggregate some cells were considered, in witch the target can move without any need for new calculation of cell values.

Key words: *path finding*, shortest path algorithms, computer games.

---

[1] PROBIC - UNIFRA.
[2] Curso de Sistemas de Informação. UNIFRA.
[3] Orientador.

INTRODUÇÃO

Em um jogo de estratégia os fatores diversão, dificuldade e variabilidade estão diretamente relacionados com o comportamento dos seus personagens. Se os assim chamados inimigos ou obstáculos fossem imóveis e estáticos, o jogo acabaria sendo muito fácil, extremamente entediante e pouco atrativo. Neste tipo de jogos, a função dos personagens e dos obstáculos a serem transpostos e derrotados é fundamental para a diversão proporcionada pelo jogo.

Um dos pontos de maior relevância recai sobre o comportamento dos personagens, amigos ou inimigos, do jogo. Eles não podem ser estáticos ou sem ação, mas precisam apresentar um comportamento capaz de passar ao usuário a idéia de que eles possuem um pensamento próprio e independente da vontade do usuário.

Para que o personagem apresente este realismo, um dos processos que devem estar incluídos no jogo é a capacidade de escolher o melhor e mais rápido caminho para chegar ao seu objetivo, e então executar a ação para o qual ele foi programado.

O objetivo deste trabalho de graduação é apresentar, discutir e solucionar o problema da escolha do melhor caminho que os personagens deverão percorrer para alcançar seu objetivo. Este caminho será definido por meio de algoritmos de escolha de caminho, também denominados algoritmos de *Path Finding.*

Algoritmos de *Path-Finding* consistem em algoritmos para obtenção de um melhor caminho para o personagem percorrer de acordo com ocupação do mapa por obstáculos e da localização do alvo a ser alcançado. Este caminho servirá para alcançar dois tipos de objetivo:

a) Chegar a um local qualquer, ou seja, a um alvo estático.

b) Chegar a um personagem, ou seja, um alvo móvel.

Estes algoritmos devem levar em conta tanto o tamanho do cenário a ser percorrido, como também encontrar uma forma de não exigir muito do processador que está rodando o jogo.

Neste trabalho, serão apresentadas e discutidas noções e idéias sobre algoritmos de *Path Finding*, seus problemas e complexidades. A partir disso, será demonstrado como foi escolhido e desenvolvido o algoritmo implementado para deslocamento em direção a alvo móvel. Serão apresentados também como se encontraram soluções para que o processamento deste algoritmo não exigisse sobremaneira a capacidade do processador. Os critérios de valores para os cálculos efetuados para escolha do menor e melhor caminho foram retirados da obra de BALCH(199-), por serem cálculos sim-

ples e por satisfazerem os objetivos do algoritmo. Também se deve ressaltar que os critérios de como seria feita a definição das posições dos obstáculos foram determinadas levando em conta outros trabalhos de formandos na área de editoração de cenários para jogos de estratégia.

METODOLOGIA

Para realização deste estudo, foram utilizados diversos trabalhos na área do estudo em questão, obras estas escolhidas por apresentarem idéias e explicações sobre maneiras de desenvolvimento do trabalho.

Após estudo e análise das obras, partiu-se então para o desenvolvimento dos algoritmos, e à medida que se encontravam problemas que fariam com que o algoritmo não funcionasse corretamente, ou que se encontravam métodos que otimizassem o encontro do melhor caminho de uma forma mais simples, o algoritmo inicial sofreu diversas modificações visando ao seu aperfeiçoamento.

Estabelecido o algoritmo, a seguir, começou a fase de implementação. A linguagem de programação *Delphi* (CANTÚ, 1997) foi a escolhida para a implementação, pelo fato de ser a que o autor melhor domina, pois se poderia ter escolhido outra linguagem.

Dos componentes do *Delphi*, poucos foram utilizados para desenvolvimento do algoritmo, a não ser para demonstrar os valores atribuídos para o *Grid*, e para demonstrar que o caminho a ser percorrido era realmente o menor. Para demonstrar o *Grid* na tela foi utilizado o componente *StringGrid*. Foi utilizado um *Button* para atribuir as células do *StringGrid* os valores armazenados na matriz de valores. Outro *Button* foi utilizado para demonstrar o caminho que seria feito pelo personagem, e um outro *Button* para simular, aleatoriamente, a movimentação do alvo.

Para representação gráfica do processo foi utilizado um componente *StrigGrid*, com quantidade de linhas e colunas referenciadas nas duas primeiras linhas do primeiro arquivo .txt carregado, em que aparecem os valores calculados para as células, bastando para isto clicar no botão Valores. O botão caminho demonstra qual foi o melhor caminho encontrado com base nos valores calculados. O botão Mover gera um número randômico, demonstrado em um *Label,* logo abaixo dele, que, de acordo com a Tabela 1, indicará qual a direção de locomoção do alvo. Clicando-se novamente, no botão valores, aparecerão os novos valores calculados, no caso do alvo ter mudado de área. Caso contrário, os valores continuam os mesmos. Clicando-se novamente, no botão caminho, mostra-se o melhor caminho recalculado.

RESULTADOS E DISCUSSÃO

EXPLANAÇÕES SOBRE UTILIZAÇÕES PARA PATH FINDING

Já foram várias as abordagens feitas ao problema de escolha de caminho feitas durante os últimos anos. Várias propostas surgiram desde então, decorrentes das diferentes áreas em que o problema poderia ser encontrado. Isso se deve às diferentes utilidades destes algoritmos.

No caso específico deste trabalho, estes algoritmos serão criados para utilização em um editor de jogos de estratégia, ajudando a definir um pouco sobre o comportamento dos personagens envolvidos no enredo do mesmo.

Mas estes algoritmos podem ser utilizados para implicações muito mais sérias e de muito mais complexidade. Um exemplo seria o desenvolvimento de ações nos mais diferentes tipos de simuladores, nas quais seria necessária Inteligência Artificial nos elementos pensantes envolvidos na simulação, tal como um exército numa simulação de guerra, para definir a melhor estratégia a ser adotada para o ataque ou a defesa.

A escolha do *melhor* caminho engloba em seu aspecto também a escolha do *menor* caminho, ou seja, elaborar um algoritmo que, de alguma forma, seja capaz de calcular um caminho com a menor distância até o objetivo a ser alcançado, seja ele móvel ou estático.

É necessário deixar claro que o algoritmo de escolha de caminho não engloba todos os aspectos necessários para o realismo dos personagens, mas é um dos principais componentes para definição do comportamento deles, por ser responsável por sua rapidez, agilidade e objetividade.

PROBLEMAS ENCONTRADOS NO DESENVOLVIMENTO E APLICAÇÕES DE PATH FINDING

Da mesma forma que os algoritmos de *Path Finding* são de uma utilidade de imensa importância, eles também geram uma série de cuidados especiais que devem ser tomados quando são desenvolvidos.

Muitas vezes o algoritmo funciona, não apresenta erros e devolve o resultado principal esperado, mas sua complexidade é tal que somente uma máquina muito poderosa é capaz de executá-lo com a rapidez necessária para imprimir realismo na aplicação em que está sendo utilizada.

Se o alvo ou objetivo desejado é móvel, ou seja, sua posição também varia, como no caso do personagem controlado pelo usuário de um jogo de estratégia, a complexidade do algoritmo aumenta ainda mais, devido ao objetivo não possuir um comportamento pré-definido, movendo-se de acordo

com o desejo da pessoa que o utiliza. Neste caso, para que o *Path Finding* gere caminhos para cada nova posição do objetivo, é necessária uma releitura de todo o ambiente do jogo cada vez que o alvo se movimentar, o que é algo muito dispendioso em termos de processamento.

CONSIDERAÇÕES SOBRE O AMBIENTE DO JOGO

Como ponto inicial para desenvolvimento do algoritmo, é necessária uma breve explicação de como será constituído o ambiente do jogo.

Seguindo o conceito exposto por JÖNSSON(199-), o primeiro passo é dividir o infinito número de lugares existentes em um espaço para um número finito de pontos. Para facilitar a visualização, estes espaços são convencionalmente representados por quadrados regulares, todos do mesmo tamanho. Estes quadrados são chamados de *Tiles* ou Células, as quais terão algum preenchimento caso este esteja ocupado, quer por um obstáculo imóvel quer por um personagem.

Um *Tile* é o espaço mínimo ocupado por um personagem ou obstáculo. Isto significa que é impossível para um personagem ou obstáculo ocupar apenas parte de um *Tile*. Mesmo que, no mudo real, o seu tamanho seja menor que o tamanho do *Tile*, no jogo, ele será considerado como ocupante de um *Tile* inteiro.

Quanto menor o *Tile*, mais próxima a representação fica do mundo real. Porém, quanto menor o *Tile*, maior a quantidade de memória necessária para armazenar a configuração do ambiente e maior a quantidade de processamento para calcular o melhor caminho.

O objetivo do algoritmo desenvolvido é fazer com que os personagens do jogo consigam alcançar um ponto qualquer do cenário, ou alcançar a entidade representada pelo personagem utilizado pelo usuário. Este personagem irá se mover pelos *Tiles,* conforme a vontade do usuário. Cada um desses *Tiles* receberá uma coordenada X, Y. No caso dos *Tiles* serem ocupados por obstáculos, as células preenchidas deverão ser aquelas intransponíveis para os personagens, e que deverão ser contornadas e superadas. Nas figuras 1 e 2, é possível visualizar um *Grid* ocupado por obstáculos.

ATRIBUIÇÃO DE UM VALOR A UMA CÉLULA

O algoritmo se baseia no método para encontrar caminhos para atingir alvos fixos proposta por BALCH(199-). Neste método, cada célula no *Grid* recebe um valor que é utilizado para fazer uma estimativa do caminho mais curto a se percorrer daquela distância até a meta. Estes valores aumen-

tam ou diminuem de acordo com a proximidade do objetivo. Quanto mais distante dele, maior o valor da célula, e quanto mais próximo menor. O valor 0 (zero) é atribuído a célula que contém o alvo a ser atingido.

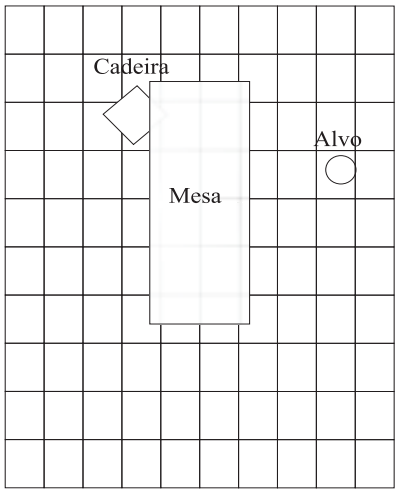

Figura 1 - Grid com obstáculos.

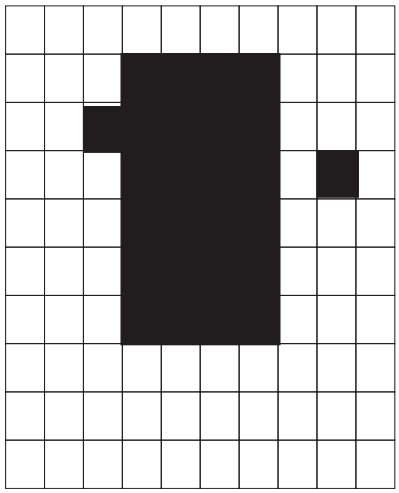

Figura 2 - Grid com células preenchidas.

Inicialmente, o programa atribui os valores às células com valores constantes, como os obstáculos e a célula que contêm a posição inicial do alvo. O valor atribuído à célula alvo sempre será 0.0, e como a navegação pelos obstáculos não é desejada, é atribuído a eles um valor extremamente grande. No caso do algoritmo em questão, este valor é de 32000, um número relativamente alto. Depois de atribuir estes valores iniciais, o algoritmo percorre o *Grid* atribuindo valores às células. O valor de cada célula que não está ocupada por obstáculos ou pelo alvo é computado verificando as células vizinhas que já tenham tido seus valores calculados, e usando uma estimativa do custo para se viajar de cada célula adjacente até a corrente. Para as células adjacentes laterais a estimativa é de 1.0, e para as adjacentes diagonais é de $\sqrt{2}$. Então, o valor da célula é, na verdade, o valor da adjacente mais o valor do deslocamento. É possível visualizar como é feito o cálculo na figura 3, considerando a célula alvo como sendo a central:

$\sqrt{2}$	1.0	$\sqrt{2}$
1.0	0.0	1.0
$\sqrt{2}$	1.0	$\sqrt{2}$

Figura 3. Cálculo dos valores.

Utilizando o cenário representado na figura 2, uma estimativa de valores para as células fica representada conforme tabela 1.

Tabela 1 - Estimativa de valores.

8.66	7.66	6.66	5.66	5.24	4.83	4.41	4.10	4.00	4.10
8.24	7.24	6.24	**32000**	**32000**	**32000**	**32000**	3.41	3.00	3.41
8.66	7.66	**32000**	**32000**	**32000**	**32000**	**32000**	2.41	2.00	2.41
9.07	8.07	7.55	5.66	**32000**	**32000**	**32000**	1.41	1.00	1.41
9.49	9.07	9.49	9.24	**32000**	**32000**	**32000**	1.00	**0.00**	1.00
10.49	10.07	9.66	8.75	**32000**	**32000**	**32000**	1.41	1.00	1.41
10.66	9.66	8.66	8.24	**32000**	**32000**	**32000**	2.41	2.00	2.41
10.24	9.24	8.24	7.24	4.83	3.95	3.41	3.41	3.00	3.41
9.83	8.83	7.83	6.83	5.83	4.83	4.41	4.41	4.00	4.41
10.24	9.24	8.24	7.24	6.24	5.83	5.41	5.41	5.00	5.41

Supondo uma visualização em três dimensões para o *Grid* com os valores tem-se (Figura 4):

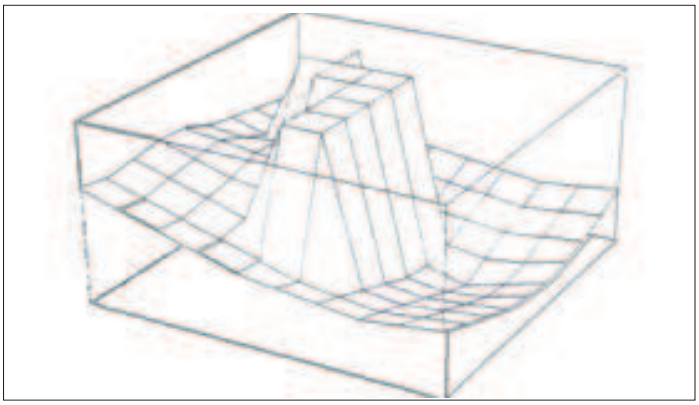

Figura 4 - Visualização 3D para os valores.

Observando esta figura, retirada de BALCH(199-), vemos os obstáculos como sendo os pontos mais altos do gráfico, e o alvo como sendo o mais baixo. Assim o caminho flui naturalmente, como se soltássemos uma bola no local de partida. Ela iria descer procurando sempre o caminho mais suave e natural para chegar ao objetivo.

PROPOSTA DO ALGORITMO

O objetivo principal é desenvolver um algoritmo simples, porém completo, de boa qualidade e, principalmente de boa funcionalidade, para ser utilizado no tipo de aplicação desenvolvida, que é a obtenção do melhor caminho dentro de um *Grid*.

O *software* desenvolvido carrega um arquivo .Txt em um componente chamado Memo1. Esse arquivo txt carregado deve conter, nas suas duas primeiras linhas, o tamanho do *Grid* que o jogo possui. Nas linhas subseqüentes, estarão informadas as posições em que são encontrados obstáculos. Para fins de funcionalidade, foi padronizado que o número 0 (zero) serviria para indicar que a célula não possui obstáculo, e o número 1 (um) caso ela esteja ocupada com um obstáculo. Com o uso de laços de repetição, os valores de cada linha do Memo1 são armazenados em uma matriz do tamanho estipulado nas duas primeiras linhas do arquivo, na seguinte ordem: posição (0,0), posição (0,1). As posições ocupadas por obstáculos receberão o valor de 32000, por se tratar de um número considerado alto, e as outras posições receberão o valor –1 (menos um), servindo-nos para calcular o melhor caminho posteriormente. Idéias similares a estas são encontradas no trabalho de JUNIOR(2000).

O *software* carrega também outro arquivo .Txt, este contendo a posição inicial do alvo. Para esta posição será atribuído o valor 0 (zero). Depois de localizado o objetivo, deve-se atribuir valores crescentes, de acordo com a distância, para as células a partir da que abriga nosso alvo. Não se pode esquecer que as células que possuem obstáculos que não podem ser ultrapassados já receberam um valor extremamente alto para que não sejam escolhidos em algum caminho. O personagem chegará até o objetivo apenas percorrendo as células procurando valores menores entre as células adjacentes a que ele está no momento. Pode haver mais de uma com valor menor do que a que ele está no momento, e ele escolhe a menor destas.

As idéias serão mais bem demonstradas em de um algoritmo desenvolvido em português estruturado, como no modelo abaixo:

Algoritmo de *Path Finding*
Realizar contagem dos *Tiles*.
Tiles contados.
Atribuir valor alto para células com obstáculos.
Valor.Célula_obstáculo = 32000.
Localizar o alvo.
Alvo localizado.
Atribuir valor zero a célula alvo.
Valor.Celula_alvo = 0.
Atribuir valores crescentes para células a partir da Célula_alvo.
Valores atribuídos.
Achar menor caminho até o alvo.
Fim do algoritmo**.**

Esta é a idéia básica para o algoritmo de *Path Finding*, que funcionaria muito bem no caso de objetivos fixos.

Cabe aqui explicar como é efetuado o processo de cálculo dos valores crescentes a partir da célula alvo. Um laço de repetição percorre todas as células vizinhas a ela, e caso o valor delas seja diferente de 32000, que seria uma célula que contém obstáculo, e também caso o valor dela seja igual a –1, valor esse atribuído inicialmente às células "livres", será calculado o valor para estas células que fazem vizinhança com a célula alvo, conforme demonstrado na figura 3. Com esses valores calculados, o laço então parte para as células que fazem vizinhança com essas já calculadas. Então, para cada célula, é verificado qual o menor valor entre as células que estão "em volta" dela e que já tenham recebido valor, senão optaria sempre pela de valor –1, e então é calculado o valor dela própria. Segue abaixo a ordem como os cálculos ocorrem, sendo representado 1º, para o primeiro laço, e 2º, para o segundo laço (Figura 5):

2º	2º	2º	2º	2º
2º	1º	1º	1º	2º
2º	1º	0.0	1º	2º
2º	1º	1º	1º	2º
2º	2º	2º	2º	2º

Figura 5 - Funcionamento do laço de repetição.

O laço deve fazer esse processo para todo o *Grid* até toda a matriz estar preenchida. Neste ponto verifica-se que o processo, nas células que seriam as "bordas" do *Grid*, tentava continuar utilizando o laço no qual não existiam mais células, podendo interferir nos cálculos, e retornar-nos valores não corretos. Devido a esse fato, resolveu-se acrescentar mais duas linhas e duas colunas no tamanho inicial do *Grid* dado pelo primeiro arquivo de texto carregado. Assim, se tomam as linhas e colunas que constituem as bordas da matriz e se atribui a elas o mesmo valor atribuído às células obstáculos. Procedendo desta maneira, as células que eram as bordas antes, e cujo cálculo de valores devem ser realizados, nunca tentam calcular seu valor, referenciando-se a células mais externas, pois seu valor é extremamente alto.

O processo de verificação do menor caminho até o alvo procede de forma semelhante ao cálculo dos valores do *Grid*. Partindo-se de um ponto de origem, predeterminado, no nosso caso a posição (1,1), faz-se uma varredura das células vizinhas a ela, verificando qual a que possui menor valor.

As coordenadas da célula de menor valor são armazenadas em uma outra matriz, e então, a partir desta célula se repete o processo de verificar qual de suas vizinhas possui o valor mais baixo. O processo se repete até que a célula de menor valor seja a célula alvo, ou seja, a de valor zero.

Foram criadas duas *functions*, uma com o nome de Vizinhos, que calcula os valores das células, e uma chamada movealvo para calcular qual o melhor caminho.

Um problema para o algoritmo ocorre no caso do objetivo ser um alvo móvel. Isso vem a ser o principal enfoque do algoritmo e a principal preocupação, pois a meta a ser cumprida é achar o menor caminho até o inimigo.

Uma das soluções que poderiam ser empregadas neste caso seria, para cada vez que o nosso alvo se movesse, e conseqüentemente se deslocasse de uma célula para outra, ocorresse um recálculo e uma reatribuição de valores para as células componentes do *Grid*. As únicas células que não necessitam de uma reatribuição de valores seriam as que estão ocupadas por obstáculos, pelo fato de seu valor ser uma constante, e as que são as bordas falsas do *Grid*. O exemplo de como ficaria o algoritmo está exposto abaixo:

Algoritmo de *Path Finding*
Realizar contagem dos *Tiles.*
Tiles contados.
Atribuir valor alto para células com obstáculos.
Valor.Célula_obstáculo = 32000.
Processo de cálculo:
Localizar o alvo.
Alvo localizado.
Atribuir valor zero a célula alvo.
Valor.Celula_alvo = 0.
Atribuir valores crescentes para células a partir da Célula_alvo.
Valores atribuídos.
Achar menor caminho até o alvo.
Fim do Processo de cálculo.
Se alvo se locomover, então,
Repete Processo de cálculo.
Fim do algoritmo.

Como se pode notar, este algoritmo, apesar de resultar em um caminho, acaba se tornando muito dispendioso em termos de processamento da máquina. Por se tratar de um jogo de estratégia em que o nosso alvo será um personagem comandado pelo usuário, é fator certo que esse personagem se

locomoverá, quase que permanentemente, durante todo o transcorrer do jogo. Devido a esse fato, o recálculo dos valores das células acaba sendo necessário por um número enorme de vezes durante o jogo. Supondo que, se a cada movimento do alvo fosse necessário efetuar a principal parte do algoritmo novamente, que é o cálculo dos valores, provavelmente, o sistema acabará ficando muito lento e pesado, por também ter de coordenar elementos gráficos do jogo.

A solução mais viável para o caso de objetos móveis e que parece ser a que melhor soluciona o problema gerado pela inconstante, porém certa, mobilidade no alvo, é de dividir-se o *Grid* em áreas, agregando um determinado número de células, conforme na figura 6.

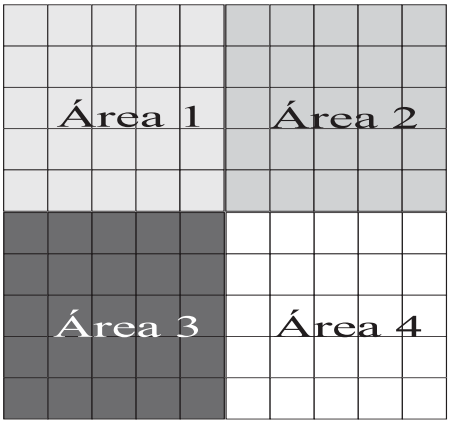

Figura 6 - Divisão em áreas.

O tamanho destas áreas variará de acordo com a quantidade de células que componham o *Grid*, e é necessário ressaltar que a definição do ambiente (onde estão os obstáculos) não foi a preocupação deste estudo, e que o tamanho da área deve ser definido pelo programador do jogo, em função da mobilidade do objetivo.

Com esta divisão procura-se traçar um primeiro objetivo para se evitar o cálculo repetitivo a todo momento. Após a localização do alvo, cálculo dos valores das células, e determinação do menor caminho inicial, realiza-se um cálculo para definir quais os limites da área no qual se encontra o alvo. Este cálculo é realizado em uma *procedure* chamada Área. Então, quando ocorre a movimentação do alvo, se ele se mantiver dentro dos limites máximos e mínimos da área, não ocorre o recálculo dos valores do *Grid*, permanecendo o caminho previamente calculado como aquele a ser seguido. No caso do alvo se deslocar para uma célula fora da área em que ele se encontra, é chamado novamente o processo de cálculo dos valores do *Grid*.

Para simular a movimentação do alvo, foi utilizado um componente *Button*, que gera um número randômico entre 0 e 8, estabelecendo assim, de

acordo com o número gerado, qual a direção do deslocamento do alvo. Considerando a posição do alvo como sendo as coordenadas X e Y, as direções foram definidas, conforme a Tabela 2.

Tabela 2 – Valores gerados aleatoriamente e direções calculadas para cada um.

Número gerado	Cálculo da direção
0	X, Y
1	X-1, Y-1
2	X, Y-1
3	X+1, Y-1
4	X+1, Y
5	X+1, Y+1
6	X, Y+1
7	X-1, Y+1
8	X-1, Y

Este *Button* chama, caso o deslocamento não for para uma célula de valor igual a 32000, a *Procedure* Movimentar, a qual, por sua vez, chama a *Procedure* Área, que verifica os limites da área em que o alvo está, e se a movimentação o leva-lo-á para uma área diferente. No caso de mudança de área ocorrer, então haverá todo o recálculo dos valores das células do *Grid*.

A partir do momento em que o personagem entra na mesma área do objetivo, são recalculados os valores a cada movimento.

Todo este processo, mesmo sendo um pouco mais complexo de se desenvolver, será muito proveitoso, pois diminuirá em muito o tempo e o trabalho do processador em calcular e recalcular os valores. Isto é mais bem ilustrado conforme exemplo abaixo:

Algoritmo de *Path Finding*
Realizar contagem dos *Tiles*.
Tiles contados.
Atribuir valor alto para células com obstáculos.
Valor.Célula_obstáculo = 32000.
Processo de calculo:
Localizar o alvo.
Alvo localizado.
Atribuir valor zero a célula alvo.
Valor.Celula_alvo = 0.
Atribuir valores crescentes para células a partir da Célula_alvo.
Valores atribuídos.

Achar menor caminho até o alvo.
Locomoção realizada.
Calcular limites da área.
    Se alvo se locomover dentro da área então
        Valores permanecem os mesmos.
    Senão,
        Se alvo se locomoveu para outra área então
            Recalculam-se valores de todas as células do *Grid*;
Recalcular menor caminho.
Fim se;
    Fim se;
    Interceptar o alvo.
Fim do algoritmo.

O algoritmo apresentado acima consegue atingir o objetivo do estudo em questão, por conseguir evitar o recálculo dos valores de todo o *Grid* a cada movimento do alvo.

CONCLUSÕES

O aperfeiçoamento do algoritmo à medida que foram surgindo os problemas, verificando-se os pontos falhos, permitiu que fossem feitos vários testes para averiguar se o comportamento do algoritmo e os valores retornados por ele eram corretos. Portanto, apenas se desenvolvendo uma aplicação é possível localizar os pontos a se aperfeiçoar.

Baseado nas experiências ocorridas do desenvolvimento do algoritmo, pode-se concluir que a idéia de atribuir valores crescentes para as células que compõem o *Grid*, a partir da célula em que se encontra o alvo, funciona de forma extremamente satisfatória, precisando-se tomar alguns cuidados para se certificar dos cálculos estarem corretos. Os valores que servem para o cálculo também se demonstraram perfeitamente funcionais.

Concluiu-se, também, que a divisão do cenário em áreas parece ser a maior contribuição que este trabalho pode trazer em termos de novidade, tendo em vista o fato de que, em nenhuma bibliografia pesquisada, foi encontrada alguma referência ao assunto. Principalmente no caso do *Grid* ser composto de muitos *Tiles*, esta proposta acabará poupando ao processador ter que refazer os cálculos a cada movimento do personagem, otimizando o funcionamento do algoritmo.

Verificando a abrangência do assunto em questão, é perfeitamente possível, com as devidas adequações, utilizar este algoritmo em outras aplicações, não só em

jogos de estratégia, mas também em softwares em que existam os fatores de desvio de obstáculos e interceptação de alvos móveis nas mais diversas áreas de estudo.

REFERÊNCIAS BIBLIOGRAFICAS

BALCH , Tucker. 199-. **Grid-based Navigation for Mobile Robots**

CANTÚ, Marco. 1997. **Dominando o Delphi 3: "a Bíblia".** São Paulo: Makron Books.

JÖNSSON, F. Markus. 199-. **An optmal pathfinder for vehicles in real-world digital terrain maps.** Tese - The Royal Institute of Science, School of Engineering Physics, Stockolm, Sweden.

JÚNIOR, Edson Prestes e Silva. 2000. **Exploração de Ambientes Desconhecidos por Sistemas Robóticos Adaptativos.**