MSX Article

Minesweeper

Índice

1 *Usando o MSX Pad como editor de Pascal para MSX*

O MSX Pad é uma excelente ferramenta para desenvolvimento de jogos em Pascal para o MSX, pois roda em PCs. Permite uma perfeita integração entre o desenvolvimento, compilação e testes em um emulador de MSX. O programa é gratuito e pode ser baixado na Internet na página do MSX Files: http://www.icongames.com.br/msxfiles/.

Após pegar o programa e instalá-lo, execute-o. Este programa só funciona no Windows 2000 e XP no modo administrador. Para criar um projeto, escolha a opção GAME e um nome para o arquivo no formato PAS, que é o código-fonte do projeto.

A figura 1 apresenta o programa. À esquerda, a hierarquia do projeto e do lado direito, o editor de textos.

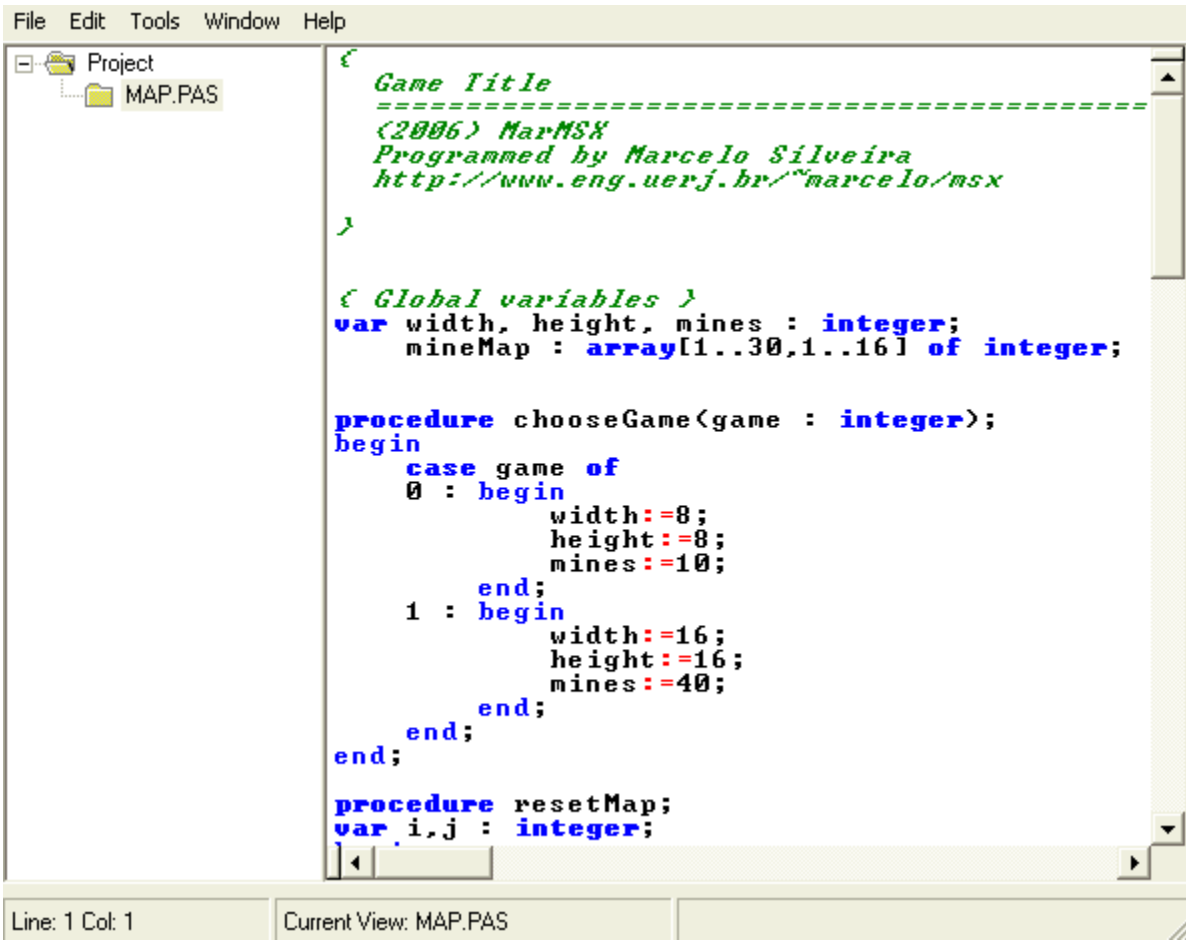

Figura 1 - MSX Pad.

Escreva seu programa e compile, para verificar os erros. Para isto, clique no menu File e em seguida em Compile ou diretamente F6. Você pode ver o resultado em um emulador de MSX. Para isto, configure em Edit, Preferences. Agora, Compile and Run ou F5.

2 *O jogo Minesweeper*

O Campo Minado (no Brasil), Minesweeper (em inglês),ou Draga-Minas (em Portugal), é um popular jogo de computador para um jogador. Foi inventado por Robert Donner em 1989 e tem como objetivo revelar um campo de minas sem detonar nenhuma mina. Este jogo de computador tem sido reescrito para as mais diversas plataformas, sendo a sua versão mais popular a que vem de raiz com o Microsoft Windows.

A área de jogo consiste num campo de quadrados retangular. Cada quadrado pode ser revelado clicando sobre ele, e se o quadrado clicado contiver uma mina, então o jogo acaba. Se, por outro lado, o quadrado não contiver uma mina, uma de duas coisas poderá acontecer:

Um número aparece, indicando a quantidade de quadrados (que se encontram adjacentes a ele) que contêm minas

Nenhum número aparece – neste caso, o jogo revela automaticamente os quadrados que se encontram adjacentes ao quadrado vazio, já que não podem conter minas

O jogo ganha quando todos os quadrados que não têm minas são revelados.

O jogador pode opcionalmente marcar qualquer quadrado que acredita que contém uma mina com uma bandeira, bastando para isso clicar sobre ele com o botão direito do rato. Em alguns casos, carregar com ambos os botões do rato num número que contenha tantas bandeiras imediatamente à sua volta quanto o valor desse número revela todos os quadrados sem bombas que se encontrem adjacentes a ele. Em contrapartida, o jogo acaba se efetuar essa ação quando os quadrados errados estiverem marcados com as bandeiras.

Algumas versões do Minesweeper ajudam o jogador, na medida em que nunca colocam uma mina no primeiro quadrado clicado (Wikipedia.org , 2006).

3 *Criando o mapa das minas*

A criação do mapa das minas pode parecer um pouco complexo, mas adotando-se a estratégia a seguir, torna-se uma tarefa fácil.

Passos:

1. Cria-se uma matriz largura x altura com o tamanho do maior tabuleiro;
2. Atribui-se o valor zero a todos os elementos da matriz;
3. Faz-se um *loop* de 1 até o número de bombas;
4. Atribui-se uma coordenada i,j com valores aleatórios, dentro da faixa do jogo;
5. Atribui o valor -1 para a bomba. Se o valor i,j fosse -1, repetir o passo 4;
6. Para a bomba corrente, notifica-se os 8 vizinhos a presença desta bomba, somando-se 1 ao valor corrente de cada célula. Se algum vizinho for bomba (valor -1), não soma.

Para exemplificar, suponha que para um tabuleiro de 8x8, conforme a tabela 1. A primeira bomba foi sorteada para a posição 3,4, conforme a tabela 2. Os vizinhos são notificados, conforme a tabela 3.

Tabela 1 - Mapa com valores iniciais.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Tabela 2 - Notificação da bomba.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	-1	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Tabela 3 - Notificação dos vizinhos.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	1	1	1	0	0	0
3	0	0	1	-1	1	0	0	0
4	0	0	1	1	1	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Se o próximo número sorteado for 3,4, deve-se repetir o sorteio até que a posição corrente seja diferente de -1. A próxima bomba sorteada for 4,5. A tabela 5 apresenta a notificação desta bomba, bem como a tabela 6 apresenta a notificação dos vizinhos.

Tabela 4 - Notificação da bomba.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	1	1	1	0	0	0
3	0	0	1	-1	1	0	0	0
4	0	0	1	1	-1	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Tabela 5 - Notificação dos vizinhos.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	0	0	1	1	1	0	0	0
3	0	0	1	-1	2	1	0	0
4	0	0	1	2	-1	1	0	0
5	0	0	0	1	1	1	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Note que a posição 4,3 não foi alterada, pois é uma bomba.

Obs: como o sistema de coordenadas da tela é x,y, e x está para j, assim como y está para i, no jogo a tabela sofre uma rotação de 90 graus, ou seja, a posição 3,4 ora citada passa a ser 4,3.

4 *Desenhando os objetos do jogo*

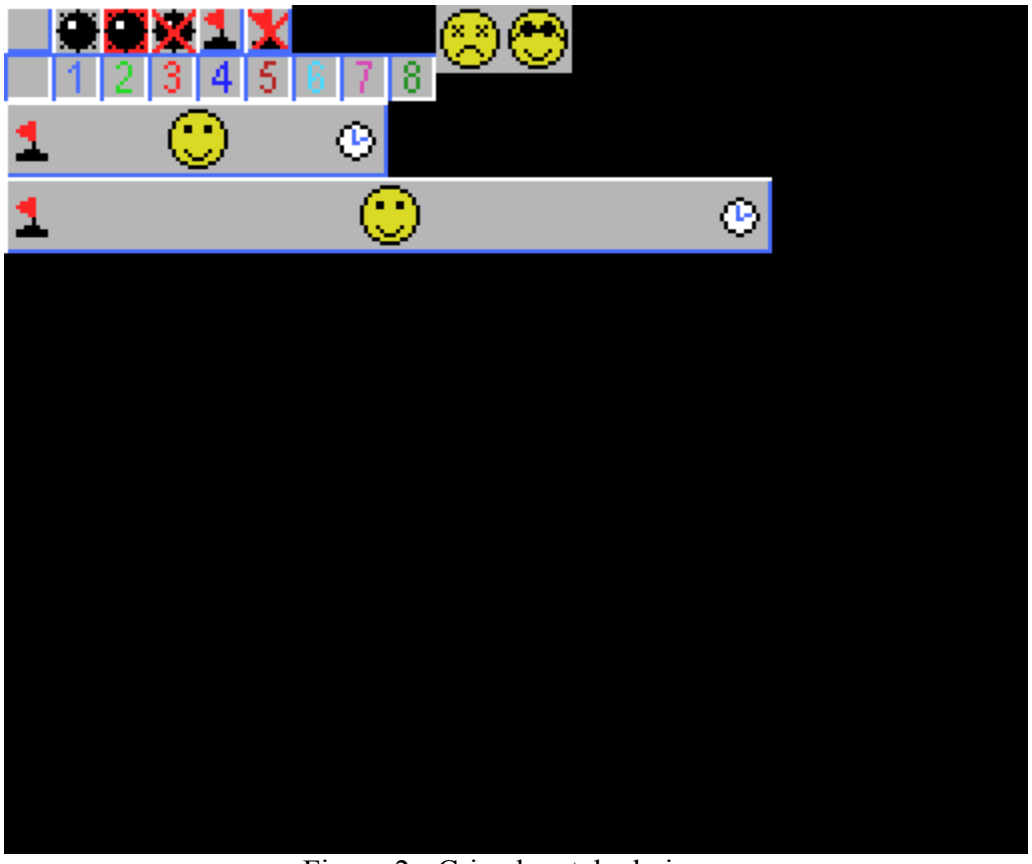

Figura 2 - Criando a tela do jogo.

Esta tela será carregada em outra página de vídeo, ou seja, ficará invisível para o jogador. Deve-se anotar as coordenadas do retângulo que envolve cada objeto e anotado para posterior uso, conforme a tabela 6.

Use o programa header.exe para extrair os 7 primeiros bytes de uma imagem dump de MSX. O programa se encontra em http://www.eng.uerj.br/~marcelo/msx/tools.

Tabela 6 - Coordenadas dos objetos.

Objeto	x_i	y_i	x_f	y_f
Caixa Alta Vazia	0	0	11	11
Bomba	12	0	23	11
Bomba Vermelha	24	0	35	11
Bandeira Errada	36	0	47	11
Bandeira	48	0	59	11
Bandeira Errada	60	0	71	11
Caixa Baixa Vazia	0	12	11	23
1 mina	12	12	23	23
2 mina	24	12	35	23
3 mina	36	12	47	23
4 mina	48	12	59	23
5 mina	60	12	71	23
6 mina	72	12	83	23
7 mina	84	12	95	23
8 mina	96	12	107	23
Header para easy	0	24	95	42
Header para hard	0	43	191	61
Sorriso	45	25	56	41
Perdeu	108	0	124	16
Ganhou	125	0	141	16

Outro ponto importante de se anotar é o alinhamento dos objetos. O jogo no modo fácil, tem o header para easy, mais uma matriz de 8x8 blocos embaixo dele. A modo de tela screen 5 do MSX possui 256 x 212 pontos. Assim, tem-se:

Jogo com 8 x 8

Header ocupa: 96 x 18
Cada bloco ocupa: 12 x 12
Todos os blocos ocupam: 96 x 96
Todo o jogo ocupa: 96 x 114
Alinhamento: 80,39

Jogo com 16 x 16

Header ocupa: 192 x 18
Cada bloco ocupa: 12 x 12
Todos os blocos ocupam: 192 x 192
Todo o jogo ocupa: 192 x 210
Alinhamento: 31,1

A biblioteca utilizada do Lammassaari apresenta erro na função Copy, para valores ímpares de x. Portanto, faça alinhamento 30,1 para o jogo 16x16.

5 *Montando o quebra-cabeças*

Uma vez definidos os desenhos, bem como anotadas as coordenadas dos objetos do jogo, podemos montar o jogo na tela.

Como o jogo possui dois modos de desenho, devemos fazer algo genérico para montar cada jogo. Então, define-se uma posição absoluta para a origem do tabuleiro em uma posição P_x, P_y, de acordo com os valores definidos no alinhamento. Cada objeto terá sua posição relativa definida. Por exemplo, o header tem sua posição relativa de H_{xr},H_{yr} de 0,0. Para saber a posição absoluta do header, faz-se $H_x = H_{xr} + P_x$ e $H_y = H_{yr} + P_y$. Assim, observa-se que ao mudar o alinhamento P_x,P_y, todos os objetos do jogo irão para a posição correta.

Para exemplificar, vamos ver quais as posições x_i,y_i do header no jogo de 8x8 ou 16x16:

Jogo de 8x8					
P_x	P_y	H_{xr}	H_{yr}	H_x	H_y
80	39	0	0	80	39

Falta definir a conversão de coordenadas relativa-absolutas para a matriz do jogo. Cada bloco ocupa 12x12 pixels, independente do modo do jogo. O sistema de coordenadas de matrizes vai de 1 a N. portanto, a posição relativa é calculada para a posição (x,y):

$M_{xr} = (x-1) * 12$
$M_{yr} = (y-1) * 12$

Agora é só criar uma tabela na memória, com todas as coordenadas dos objetos. Defina 0 para a caixa baixa vazia e de 1 a 8 os objetos com estes números. Assim, podemos montar o mapa mais facilmente.

Para um processador Z-80, uma conta de multiplicação leva muito tempo para ser processada. Então, vamos criar um offset para calcular a posição dos blocos.

A implementação do código para desenhar o tabuleiro do jogo ficará assim:

```
procedure mountTable;
var i,j, page : integer;
     offsetX, offsetY : integer;
begin
    {Copy(X1,Y1,X2,Y2,SrcPage,DestX,DestY,DestPage:integer);}
    {line_bf(x,y,x2,y2:integer;color,log_op:byte);}

    page:=0;

    { Header }
    Copy(scrCoord[12+gameType,1],scrCoord[12+gameType,2],
    scrCoord[12+gameType,3],scrCoord[12+gameType,4],
    1,
```

```
    0+gameAlign[gameType,1],0+gameAlign[gameType,2],
    page);
    { Matrix }
                              offsetY:=scrCoord[12+gameType,4]-
scrCoord[12+gameType,2]+gameAlign[gameType,2]+1;
    for i:=1 to gameBounds[gameType,1] do
    begin
      offsetX:=gameAlign[gameType,1];
      for j:=1 to gameBounds[gameType,2] do
      begin
          Copy(scrCoord[0,1],scrCoord[0,2],
         scrCoord[0,3],scrCoord[0,4],
        1,
        offsetX,offsetY,
        page);
        offsetX:=offsetX+12;
      end;
      offsetY:=offsetY+12;
    end
end;
```

O tópico a seguir apresenta como iremos mapear os objetos do jogo.

6 *Mapeando os objetos do jogo*

Para mapear os objetos do jogo, iremos criar uma tabela com as coordenadas dos objetos na imagem que foi criada. No Pascal, somente o tipo de dado constante permite atribuição na declaração. Assim, temos:

```
const   scrCoord   :   array[-1..18,1..4]   of   integer   =
     ((12,0,23,11),
     (0,12,11,23),
     (12,12,23,23),
     (24,12,35,23),
     (36,12,47,23),
     (48,12,59,23),
     (60,12,71,23),
     (72,12,83,23),
     (84,12,95,23),
     (96,12,107,23),
     (0,0,11,11),
     (24,0,35,11),
     (36,0,47,11),
     (48,0,59,11),
     (0,24,95,42),
     (0,43,191,61),
     (45,25,56, 41),
     (108,0,124,16),
     (123,0,141,16)
     (60,0,71,11));

     {Valores}

-1   Bomba                  12   0    23   11
0    Caixa Baixa Vazia      0    12   11   23
1    1 mina                 12   12   23   23
2    2 mina                 24   12   35   23
3    3 mina                 36   12   47   23
4    4 mina                 48   12   59   23
5    5 mina                 60   12   71   23
6    6 mina                 72   12   83   23
7    7 mina                 84   12   95   23
8    8 mina                 96   12   107  23
9    Caixa Alta Vazia       0    0    11   11
10   Bomba Vermelha         24   0    35   11
11   Bandeira Errada        36   0    47   11
12   Bandeira               48   0    59   11
```

```
13   Moldura para easy        0    24   95   42
14   Moldura para hard        0    43   191  61
15   Sorriso                  45   25   56   41
16   Perdeu                   108  0    124  16
17   Ganhou                   123  0    141  16
18   Bandeira Errada          60   0    71   11
```

Consultando o capítulo 5, podemos observar que o comando copy usa as coordenadas x_i, y_i, x_f, y_f dos objetos no desenho original.

A imagem de desenho irá ficar localizada na página 1 da screen 5 do MSX.

Quando referenciamos scrCoord[-1,1], o valor x_i do desenho da bomba é retornado.

7 *O algoritmo flood-fill*

Quando o jogador clica em uma casa que está vazia, ou seja, o número de bombas é zero, todas as casas em torno desta casa são abertas, até que se encontre uma casa não vazia. O algoritmo que preenche uma região, assim como as ferramentas de pintura para programas gráficos, é o flood-fill (Wikipedia: http://en.wikipedia.org/wiki/Flood_fill).

Para pintarmos uma região de um desenho 2D, faz-se os seguintes passos:

```
Flood-fill (nó, cor_alvo, cor_a_substituir):
 1. Se a cor do nó for diferente à cor de alvo, retorne.
 2. Defina a cor do nó para a cor a substituir.
 3. Aplique a função flood-fill (um passo para oeste do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para leste do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para norte do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para sul do nó,
cor_alvo, cor_a_substituir).
 4. Retorne.
```

Este algoritmo terá que ser adaptado para a questão do jogo de minas, pois a diferença é que no jogo, as casas limítrofes da casa vazia são abertas também. Assim:

```
Flood-fill (nó, cor_alvo, cor_a_substituir):
 1. Defina a cor do nó para a cor a substituir.
 2. Se a cor do nó for diferente à cor de alvo, retorne.
 3. Aplique a função flood-fill (um passo para oeste do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para leste do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para norte do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para sul do nó,
cor_alvo, cor_a_substituir).
Aplique a função flood-fill (um passo para noroeste do nó,
cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para nordeste do
nó, cor_alvo, cor_a_substituir).
    Aplique a função flood-fill (um passo para sudeste do nó,
cor_alvo, cor_a_substituir).
        Aplique a função flood-fill (um passo para sudoeste
do nó, cor_alvo, cor_a_substituir).
 4. Retorne.
```

Trocando-se a linha 1 pela 2, a casa é sempre aberta, porém não dispara a função para os seus vizinhos. Outra mudança é em relação à vizinhança. Agora, dispara-se para os 8 vizinhos, pois se a casa é vazia, não há bomba ao seu redor.

Para marcar as jogadas, criam-se mais 2 matrizes: uma para indicar se a casa foi aberta ou não e a outra para indicar se o jogador marcou alguma casa com uma bandeirinha.

```
var flagMap : array[1..16,1..16] of boolean;
    clickMap : array[1..16,1..16] of boolean;
```

Para os sistemas CP/M-80 que utilizam o Pascal, no caso, o modelo do MSX, deve-se utilizar a diretiva {$A-} para fazer recursividades. O algoritmo que faz o flood fill é o seguinte:

```
{$A-}
procedure flood_fill(i,j : integer);
var x,y : integer;
     flag : boolean;
begin
    {If out of bounds, return}
    if (i<1) or (j<1) or (i>8) or (j>8) then
      exit;

    {If node is already visited, return}
    if (mapOpen[i,j]=true) then
      exit;

    {Open node}
    mapOpen[i,j]:=true;

    {If node <> empty, return}
    if (map[i,j]<>0) then
      exit;

    {Spread this to its 8 neighbors}
    for x:=-1 to 1 do
    begin
        for y:=-1 to 1 do
        begin
               if not((x=0) and (y=0)) then
                        flood_fill(i+x,j+y);
        end;
    end;
end;
```

8 *O controle do jogo*

O jogo é composto das seguintes etapas:

- Menu para escolher uma opção de jogo
- Preparação do jogo
- Rotina de controle do jogo

O menu escolhido para a ocasião é o menu de barras. Ele irá determinar se o jogo é do tipo 1, 2 ou se o jogador quer sair do jogo.

Uma vez escolhido o jogo, os seguintes passos deverão ser dados:

- Determinar o tipo de jogo, através da variável gameType.
- Carregar a rotina new game, que é responsável por rodar as rotinas de criação de mapa do arquivo map.pas e ajustar algumas variáveis. São elas:

    - chooseGame(gameType);
    - resetMap;
    - createMap;
    - flags:=mines;
    - cursorX:=1;
    - cursorY:=1;

- Passar o controle do jogo para o loop principal.

O loop principal fica a espera do jogador teclar algo. Enquanto isso o programa vai atualizando o número de bandeiras e o tempo de jogo. A função keypressed desvia o loop para seu interior, quando uma tecla é pressionada, de forma a tratar o evento.

9 *Contando o tempo de jogo*

A maneira mais eficaz de se contar o tempo decorrente de um jogo é consultando um dispositivo independente de relógio. A partir dos MSX 2, o sistema passa a contar com um sistema de relógio, independente ao processador Z-80. Assim, para se contar o tempo decorrido, basta marcar o horário inicial e depois o horário corrente. Faz-se a diferença e obtém-se o tempo decorrido.

O relógio do MSX trabalha com horas, minutos e segundos, diferente dos relógios do PC se são capazes de manipular os milésimos de segundos. Assim, podemos ter uma certa eficiência no que diz respeito aos segundos decorrentes, podendo perder a precisão entre o primeiro e o último segundo apenas, pois o sistema não nos fornece os milésimos de segundo.

Há dois caminhos para se obter a hora corrente no MSX: a função RDCLK da BIOS ou a função GTIME (&H2C) do MSX DOS. Como o jogo roda sob o DOS, vamos utilizar a última. A descrição da mesma segue abaixo[1]:

GTIME (2CH)
Função: Leitura da hora.
Setup: Nenhum.
Retorno:
H = horas;
L = minutos;
D = segundos;
E = centésimos de segundo.

Apresar do sistema possuir um retorno para centésimos de segundo, o relógio do MSX não nos fornece esta informação. Portanto, E deverá ser descartado.

Para utilizar um código em Assembly devemos usar a função inline do Pascal. Endereços de memória podem ser substituídos por nomes de variáveis.

O programa precisa de duas funções em Pascal para marcar o tempo decorrido: uma para guardar a hora inicial e outra para pegar a hora corrente e calcular o tempo decorrido.

A função que calcula o tempo decorrido utiliza somas para evitar realizar cálculos de multiplicação, muito custosos. O algoritmo completo pode ser visto no arquivo *minetime.inc*, que acompanha o jogo.

1- Fonte: MSX Top Secret 1, de Edison Moraes em http://www.msxtop.msxall.com

10 *O ranking de melhores tempos*

O ranking para o jogo de minas, com 2 níveis de dificuldade serão 2 tabelas na memória. Temos 2 tipos de dados para cada elemento da tabela: string para o nome do jogador e integer para o tempo gasto. Assim, criaremos o tipo rank:

```
Type rank = record
              name : string[80];
              time : integer;
            end;
     TName = string[80];
```

Para este jogo, criaremos uma tabela apenas com 20 posições do tipo rank;

TName é um tipo string de 80 posições para o nome da pessoa. Na verdade, iremos futuramente utilizar somente 12 caracteres para o nome, mas vamos deixar um *default* de 80.

O primeiro passo será limpar as tabelas e assinalar com o valor -1 que este espaço se encontra vazio. Então:

```
procedure clearRanking;
var i : integer;
begin
  for i:=0 to 20 do
  begin
    ranking[i].name:='';
    ranking[i].time:=-1;
  end;
end;
```

Esta tabela não é uma tabela que recebe todos os elementos de uma vez. Portanto, nenhum algoritmo de ordenação vai ser preciso. Desta forma, colocaremos os elementos um a um na tabela, se e somente se:

1- A posição corrente de comparação é vazia; ou
2- A posição corrente de comparação é maior ou igual ao valor de inserção.

Quando o jogador sair do jogo, os resultados serão perdidos! Desta forma, teremos que guardar os valores em disco. Primeiro, define-se um nome para o arquivo de saída, bem como uma variável de arquivo do tipo rank. Em seguida, o programa é apresentado.

```
const rankFile = 'minerank.dat';
var rankDisk : file of rank;
```

```
procedure saveRanking;
var i : integer;
begin
  writeln('Writing ranking to disk ...');
  Assign(rankDisk,rankFile);
  Rewrite(rankDisk);

  for i:=1 to 20 do
    Write(rankDisk,ranking[i]);
  close(rankDisk);
end;
```

Quando o jogo é carregado, necessita-se verificar se a tabela do jogo existe. Para tal, foi necessário criar uma rotina[1] que verifica a existência de um arquivo.

```
function FileExists(Name : TName) : boolean;
Var
  F : File;
begin
  {$I-}
  Assign (F,Name);
  Reset (F);
  {$I+}
  FileExists:=(IoResult=0) and (Name<>'');
  Close (F);
end;
```

Se o arquivo de ranking existir, então carregue-o. Senão, limpe a tabela.
O formato do arquivo dos dados dos melhores tempos é binário.

11 *Corrigindo o problema do random*

Para corrigir o problema do random, deve-se aumentar a quantidade de números e depois normalizar o resultado. Por exemplo:

```
function randomTime(num : integer) : integer;
var  seed, max : integer;
Begin
    randomize;

    max:=99;
    seed:=random(100);
    seed:=(seed*num) div max;

    randomTime:= abs(seed);
end;
```

12 *Créditos*

O programa MSX Minesweeper, bem como o manual foram elaborados por Marcelo Teixeira Silveira.

E-mail: flamar98@hotmail.com.
Homepage: http://www.eng.uerj.br/~marcfelo/msx