

MSX Article

MARMSX

*Desenvolvimento
do Jogo
Minesweeper*

Índice

Capítulo	Pág
1- Usando o MSX Pad como editor de Pascal para MSX	3
2- O jogo Minesweeper	5
3- Criando o menu de barras	7
4- Criando o mapa das minas	10
5- Desenhando os objetos do jogo	12
6- Montando o quebra-cabeças	14
7- O algoritmo flood-fill	16
8- O controle do jogo	18
9- Contando o tempo de jogo	20
10- O ranking de melhores tempos	21
11- Corrigindo o problema do random	23
12- Créditos	24

Versão do documento: 1.2

Data: Novembro de 2018

1

Usando o MSX Pad como editor de Pascal para MSX

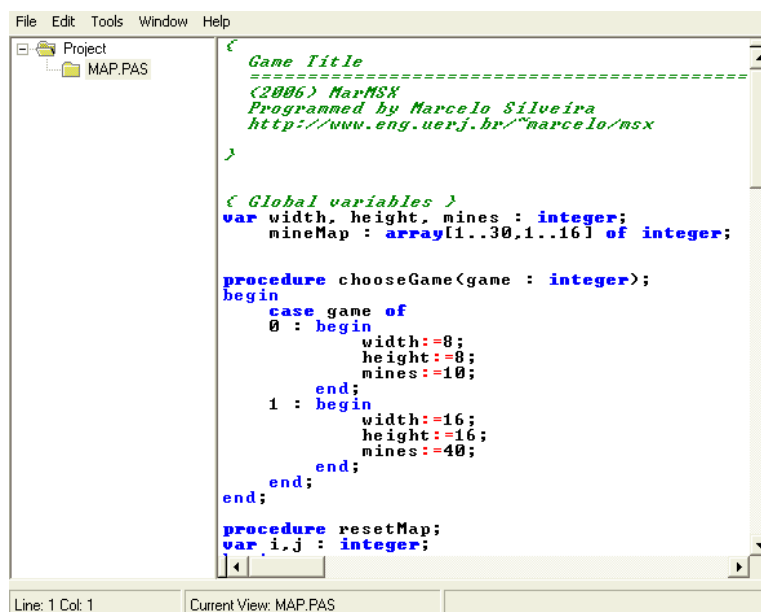
O MSX Pad é uma excelente ferramenta para desenvolvimento de jogos em Pascal para o MSX, uma vez que roda em PCs. Dessa forma, ele apresenta uma série de vantagens como o texto destacado em cores diferentes, uma compilação mais rápida do código fonte em Pascal, além de possuir um ambiente integrado para o seu projeto e a integração entre o aplicativo e um emulador de MSX, permitindo executar automaticamente o código desenvolvido.

O programa é gratuito e pode ser baixado na Internet na página da MSX Files: <http://www.icongames.com.br/msxfiles/>. Ele deverá ser instalado no sistema operacional Windows e, após instalar o programa, ele deverá ser rodado no modo administrador para funcionar corretamente.

A primeira tela que surge é um menu que pedirá ao usuário que escolha um entre quatro templates (modelos de projeto). Para criar um jogo, utilize o template “game”. Em seguida, escolher um nome para o arquivo principal do projeto “.pas”.

Cada template fornece alguns arquivos de inclusão da biblioteca do Lammassaari, que foram melhorados ou adaptados pelo autor do programa. Entretanto, é possível adicionar outros arquivos de inclusão ao projeto, desde que não se encontrem naqueles fornecidos.

A figura 1 apresenta o programa. À esquerda, a hierarquia do projeto e do lado direito, o editor de textos. Observe que o texto destaca o código em Pascal em cores temáticas.



```
File Edit Tools Window Help
Project
  MAP.PAS
  < Game Title
  -----
  <2006> MarMSX
  Programmed by Marcelo Silveira
  http://www.eng.uerj.br/~marcelo/msx
  >
  < Global variables >
  var width, height, mines : integer;
      mineMap : array[1..30,1..16] of integer;

  procedure chooseGame(game : integer);
  begin
    case game of
      0 : begin
          width:=8;
          height:=8;
          mines:=10;
        end;
      1 : begin
          width:=16;
          height:=16;
          mines:=40;
        end;
    end;
  end;

  procedure resetMap;
  var i,j : integer;
```

Line: 1 Col: 1 Current View: MAP.PAS

Figura 1.1 - MSX Pad.

Escreva seu programa e compile, para verificar os erros. Para isto, clique no menu File e em seguida em Compile ou tecla “F6”.

Você pode ver o resultado em um emulador de MSX. Para isto, configure o caminho dele em Edit, Preferences. Depois, escolha a opção “Compile and Run” ou tecla “F5”.

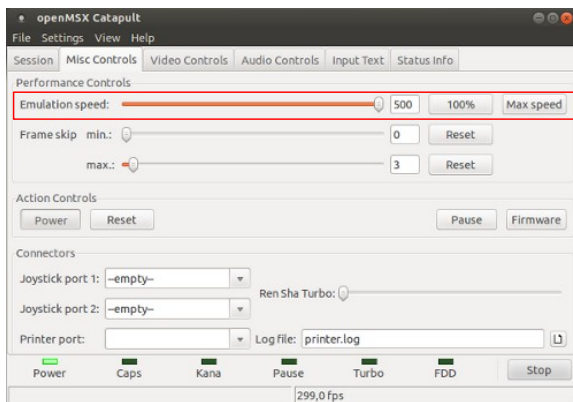
1.1. Alternativa: compilando usando um emulador de MSX

Uma alternativa ao MSX Pad é o desenvolvimento utilizando o compilador Turbo Pascal 3.3f, que é o mais moderno. Ele compila rápido e o tamanho do arquivo “.COM” gerado é bem pequeno.

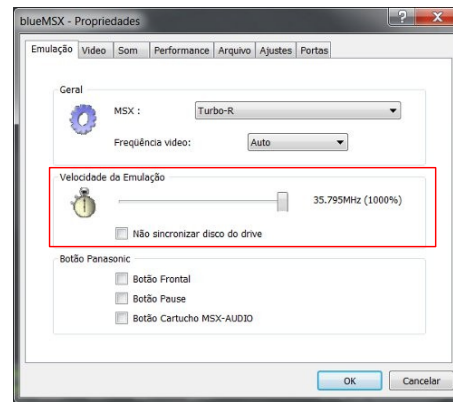
Se você quiser trabalhar em um MSX Real, utilize junto ao compilador o editor de textos Mac Program Writer (MPW) para MSX 2. Entretanto, utilizando um emulador para MSX teremos algumas vantagens descritas a seguir.

Vantagens de se utilizar um emulador:

- Intercâmbio direto de arquivos com o PC – podemos editar um arquivo no formato texto em qualquer editor do PC (Windows e Linux), salvá-lo e ter acesso a ele no emulador, uma vez que tanto o blueMSX como o openMSX acessam um diretório como se fosse um disco virtual.
- Compilação mais rápida – podemos aumentar a velocidade de emulação do MSX e obter o resultado da compilação mais rápido. O openMSX é capaz de emular até 5 vezes mais rápido e o blueMSX 10 vezes.



a) openMSX



b) blueMSX

Figura 1.2 – Alterando a velocidade de emulação.

Nota: por motivo de segurança, copie os arquivos textos do programa editado para um disquete virtual “.dsk” contendo o TP 3.3f e compile dentro desse disco virtual. EVITE compilar em um diretório montado como um disco virtual.

No emulador:

Disco a : tp.dsk

Disco b : c:\MSX\projeto

No MSX-DOS:

```
A>copy b:prog.pas a:
```

```
A>turbo prog
```

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. Ele 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 - indica 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 jogador ganha quando todos os quadrados que não têm minas são revelados.

O jogador pode opcionalmente marcar qualquer quadrado que ele acredita que contém uma mina com uma bandeira, bastando para isso clicar sobre ele com o botão direito do mouse. Em alguns casos, carregar com ambos os botões do mouse 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 (fonte: Wikipedia.org , 2006).

A Versão do jogo de Minas para o MSX

No jogo desenvolvido para o MSX, há duas versões para esse jogo: a versão mines e a versão supermines. A primeira contém a versão do jogo com 8x8 blocos e 10 bombas e outra com 16x16 blocos e 40 bombas. Já a segunda versão possui apenas a configuração de 30x16 blocos e 99 minas.



Figura 2.1 – Jogo de Minas para o MSX.

Como jogar:

- Escolha o tipo de jogo:
 - Versão mines:
 - Fácil - 8x8 com 10 bombas
 - Difícil - 16x16 com 40 bombas
 - Versão supermines:
 - Única opção - 30x16 com 99 bombas
- As setas movimentam o cursor sobre os blocos.
- A tecla “esc” abandona o jogo.
- A tecla “Z” detona o bloco sob o cursor, liberando um espaço vazio, um número ou explodindo uma bomba.
- A tecla “X” marca ou desmarca o bloco com uma bandeira.
- A tecla “C” quando clicado sobre um número abre seus 8 vizinhos.
 - Somente se nessa vizinhança existir pelo menos uma bandeira.
- Tecla “enter” quando ganhar o jogo, para abrir o menu inicial.

Configurações mínimas:

- MSX 2
- MSX-DOS 1
- 64 KB RAM
- 128 KB VRAM

O jogo é capaz de marcar o tempo gasto para abrir as minas e também salva em disco os recordes do jogador (ranking).



Figura 2.2 – Recordes.

3

Criando o menu de barras

A criação de um menu de barras é uma tarefa simples no MSX 2, uma vez que é possível definir uma cor de fundo para o texto escrito. Por exemplo, o seguinte código em Basic irá escrever um texto branco com o fundo verde, em uma tela preta.

```
10 COLOR 15,0,0
20 SCREEN 5
30 OPEN"GRP:"AS#1
40 COLOR 15,12
50 PSET(10,10),12:PRINT#1,"Mines"
60 GOTO 60
```

O resultado pode ser visto na figura 3.1.

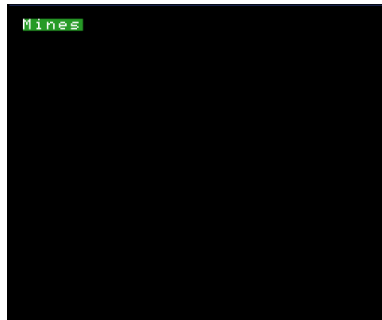


Figura 3.1 – A cor de fundo em textos do MSX 2.

É possível criar um menu, simplesmente trocando a cor de fundo da opção corrente através desse recurso. Evidentemente, as opções antigas deverão ser recuperadas para as cores originais.

Porém, o retângulo da área de fundo dos caracteres não pode ser modificado. Dessa forma, não podemos ter um retângulo maior sob a opção corrente. Nesse caso, devemos apelar para outro recurso do MSX 2: o desenho com o uso de operações lógicas.

Podemos então criar um retângulo do tamanho desejado e copiá-lo sobre a opção corrente, utilizando uma operação lógica que misture as cores de origem e de destino. Obviamente, essa opção limita a combinação de cores do menu, uma vez que o resultado advém de uma combinação de cores.

Primeiramente, vamos criar o menu em Basic para a melhor compreensão do código. Ele consiste em um retângulo preenchido na cor azul, que servirá de fundo para o menu. A borda do retângulo é desenhada em branco para dar destaque. As opções são escritas dentro desse espaço também na cor branca.

```

10 COLOR 15,0,0
20 SCREEN 5
30 OPEN"GRP:"AS#1
40 COLOR 15,4,0
50 LINE(90,90)-(190,135),4,BF
60 LINE(90,90)-(190,135),15,B
70 PSET(95,95),4:PRINT#1,"Mines Easy"
80 PSET(95,105),4:PRINT#1,"Mines Hard"
90 PSET(95,115),4:PRINT#1,"Records"
100 PSET(95,125),4:PRINT#1,"Exit"
110 GOTO 110

```



Figura 3.2 – O menu desenhado.

Em seguida, o menu é copiado para a página 1, para posterior recuperação.

Para desenhar o retângulo da opção corrente, deve-se desenhá-lo utilizando uma operação lógica, senão, o texto seria sobreposto pelo desenho.

Modificando o programa anterior, iremos assinalar uma opção com o retângulo. A variável “I” controla a opção corrente do menu.

```

110 I=1
120 LINE(93,93+(I-1)*10)-(175,95+(I-1)*10+8),7,BF,OR
130 GOTO 130

```



Figura 3.3 – O menu desenhado e a opção marcada.

Cada vez que as teclas “para cima” ou “para baixo” são pressionadas, a cópia do menu original que está na página 1 é copiada de volta para a página 0. Assim, a opção é desmarcada e o menu está pronto para receber a nova opção.

A operação lógica OR combina a cor ciano 7 do retângulo com o fundo azul 4 e o texto branco 15. Assim temos:

Fundo:		Texto:
4 - 00000100		15 - 11111111
7 - 00000111	OR	7 - 00000111
-----		-----
7 - 00000111		15 - 11111111

As cores resultantes da operação lógica OR são o ciano 7 para o fundo e o branco 15 para o texto.

A seguir, será apresentado o código em pascal do menu principal.

```
while ord(c)<>255 do
begin
  Color(15,4,0);
  line_bf(90,90,160,135,4,0);
  line_b(90,90,160,135,15,0);
  Print(95,95,15,'Mines Easy');
  Print(95,105,15,'Mines Hard');
  Print(95,115,15,'Records');
  Print(95,125,15,'Exit');
  Copy(90,90,160,135,0,90,190,1);
  line_bf(93,93,155,95+8,7,2);

  i:=1;
  c:='a';
  while ord(c)<>13 do
  begin
    read(kbd,c);
    Copy(90,190,160,235,1,90,90,0);
    case ord(c) of
      31 : begin
          i:=i+1;
          if (i>4) then i:=1;
        end;
      30 : begin
          i:=i-1;
          if (i<1) then i:=4;
        end;
    end;
    line_bf(93,93+(i-1)*10,155,95+(i-1)*10+8,7,2);
  end;
  { ... }
end;
```

O código 2 aqui equivale a operação lógica OR.

4

Criando o mapa das minas

A criação do mapa das minas pode parecer um pouco complexo, mas adotando-se a seguinte estratégia, ficará mais fácil de fazer.

Passos:

1. Cria-se uma matriz de inteiros com o tamanho do tabuleiro.
2. Atribui-se o valor zero a todos os elementos dessa matriz.
3. Faz-se um *loop* de 1 até o número de bombas existentes.
4. Escolhe-se aleatoriamente uma coordenada i,j da matriz. Caso a coordenada não esteja vazia (valor 0), repita essa operação.
5. Atribui-se o valor -1 para essa coordenada, indicando que ali existe uma bomba.
6. Para essa bomba, deve-se notificar os 8 vizinhos a presença dela.
 - 6.1. Soma-se 1 ao valor corrente de cada célula.
 - 6.2. Se algum vizinho for bomba (valor -1), não soma.

Para exemplificar, vamos criar um tabuleiro de 8x8 blocos, conforme mostra a tabela 1.

	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 1 - Mapa com valores iniciais.

A primeira bomba foi sorteada aleatoriamente para a posição 3,4, conforme mostra a tabela 2.

	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 2 - Notificação da bomba.

Os vizinhos são notificados da presença dessa bomba, conforme mostra a tabela 3.

	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 3 - Notificação dos vizinhos.

Se a próxima posição sorteada também for 3,4, deve-se repetir o sorteio até que a posição corrente não seja uma bomba, ou seja, possua valor diferente de -1.

A próxima bomba sorteada foi 4,5. Não há problema se a bomba cair sobre uma região com a notificação de outra bomba. A tabela 4 apresenta a notificação desta bomba e a tabela 5 apresenta a 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

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	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

Tabela 5 - Notificação dos vizinhos.

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

Obs: como o sistema de coordenadas da tela é x,y , 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.

5

Desenhando os objetos do jogo

Todos os objetos gráficos do jogo serão desenhados em uma tela, que será carregada na página 1 do vídeo do MSX. Esta página serve de rescunho e será invisível para o usuário, uma vez que a página visível padrão é a página 0.

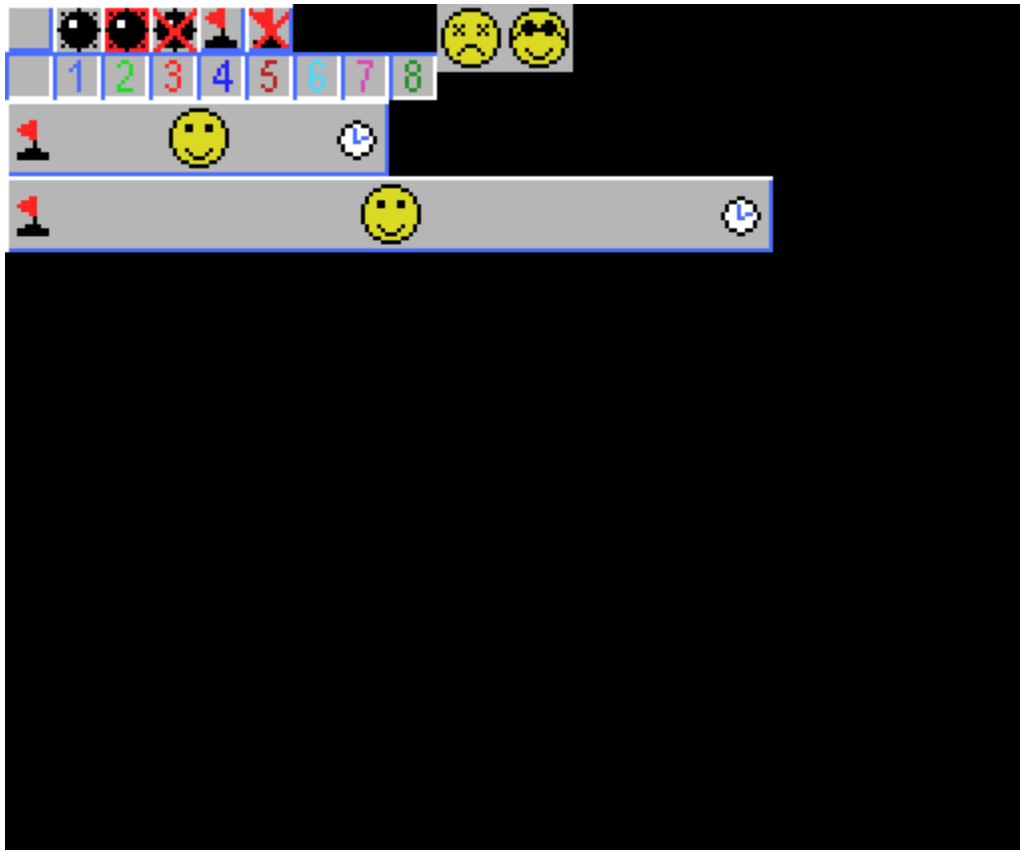


Figura 5.1 - A tela de rascunho do jogo.

Os objetos da figura 5.1 são os objetos necessários para desenhar todo o tabuleiro do jogo. Eles serão desenhados na página 0 conforme o desenrolar do jogo.

Primeiramente, deve-se anotar as coordenadas retângulo envolvente de cada objeto na tela de rascunho para que posteriormente sejam copiados para a tela do jogo, utilizando o recurso de cópia de regiões de tela do MSX 2.

A tabela 6 apresenta a coordenada do retângulo envolvente de cada objeto da tela de rascunho.

Obs: o arquivo de imagem deverá estar sem o cabeçalho de 7 bytes, comum em imagens do MSX. Para eliminar esse cabeçalho, utilize o programa *header* que se encontra na seção *tools* da página MarMSX: <http://marmsx.msxall.com>

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 minas	24	12	35	23
3 minas	36	12	47	23
4 minas	48	12	59	23
5 minas	60	12	71	23
6 minas	72	12	83	23
7 minas	84	12	95	23
8 minas	96	12	107	23
Controle do jogo para easy	0	24	95	42
Controle do jogo para hard	0	43	191	61
Sorriso	45	25	56	41
Perdeu	108	0	124	16
Ganhou	125	0	141	16

Tabela 6 - Coordenadas dos objetos.

Outro ponto importante é calcular o posicionamento do tabuleiro do jogo. Por exemplo, o jogo no modo fácil utiliza o “controle do jogo” com o tamanho menor, mais uma matriz de 8x8 blocos embaixo dele. Assim, tem-se:

Jogo com 8 x 8

Controle do jogo ocupa: 96 x 18 pixels
 Cada bloco ocupa: 12 x 12 pixels
 Todos os blocos ocupam: 96 x 96 pixels
 Todo o jogo ocupa: 96 x 114 pixels

Como o modo de tela screen 5 do MSX possui 256 x 212 pontos, temos:
 Posição x_i, y_i do jogo: $(256-96)/2, (212-114)/2 = 80,49$

O mesmo pode ser feito para o jogo no modo difícil.

Jogo com 16 x 16

Controle do jogo ocupa: 192 x 18 pixels
 Cada bloco ocupa: 12 x 12 pixels
 Todos os blocos ocupam: 192 x 192 pixels
 Todo o jogo ocupa: 192 x 210 pixels

Posição x_i, y_i do jogo: $(256-192)/2, (212-210) = 32,2$

Obs: A biblioteca utilizada do Lammassaari apresenta erro na função “Copy”, para valores ímpares de x. Desse modo, utilizar valores pares para x.

6

Montando o quebra-cabeças

Uma vez feitos os desenhos dos objetos, bem como anotadas suas coordenadas na tela, podemos montar a tela do jogo. Como o jogo possui dois modos de desenho, devemos fazer um procedimento genérico para montar cada jogo.

Primeiramente, deve-se definir a origem de coordenadas do jogo para a posição P_x, P_y calculada no posicionamento do jogo (capítulo 5). Assim, cada objeto terá sua posição ajustada para a posição relativa a P_x, P_y . Por exemplo, o controle do jogo tem sua posição relativa C_{xr}, C_{yr} em $0,0$. Para calcular sua posição real na tela, deve-se fazer o seguinte cálculo:

$$C_x = C_{xr} + P_x$$

e

$$C_y = C_{yr} + P_y$$

Fazendo desse modo, quando o valor de P_x, P_y mudar, todos os objetos do jogo irão para uma nova posição junstos. Esse sistema satisfaz o jogo de 8x8 e 16x16.

Falta definir a conversão de coordenadas para a matriz de blocos do jogo. Como cada bloco ocupa 12x12 pixels, independente do modo do jogo, e o sistema de coordenadas de matrizes vai de 1 a N , a posição relativa é calculada da seguinte forma:

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

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

Uma vez calculadas as posições relativas de um bloco, é necessário ainda calcular as suas coordenadas reais, conforme foi feito para o controle do jogo.

Para facilitar o trabalho de localização dos objetos, podemos criar uma tabela na memória com todas as coordenadas desses objetos. Defina 0 para o bloco vazio e de 1 a 8 os objetos contendo os respectivos números.

Para um processador Z80, uma conta de multiplicação leva muito tempo para ser processada. Então, vamos criar um offset para calcular a posição dos blocos.

O código que cria uma matriz com os valores das coordenadas dos objetos e também do posicionamento do jogo é apresentado a seguir.

```
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),
(40,25,56, 41), (108,0,124,16), (124,0,141,16), (60,0,71,11));

gameAlign : array[1..2,1..2] of integer = ((80,39), (30,1));
```

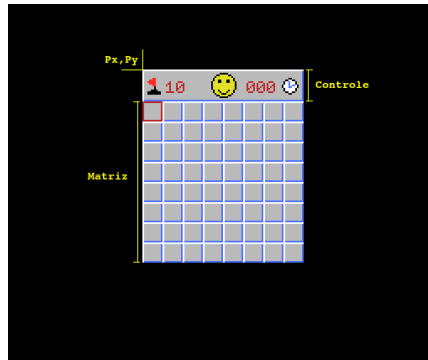


Figura 6.1. Layout do jogo na tela.

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

```

procedure mountTable;
var i,j, page : integer;
    offsetX, offsetY : integer;
begin
  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;
end;

```

Obs: as variáveis “offsetX” e “offsetY” deslocam a posição original da matriz para a célula corrente, para que não sejam feitos cálculos de multiplicação durante o desenho de cada célula e acelerar o processamento.

O tópico a seguir apresenta como são mapeados os objetos do jogo.

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 automaticamente 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. Maiores detalhes sobre esse algoritmo pode ser encontrado em Wikipedia: http://en.wikipedia.org/wiki/Flood_fill.

O algoritmo utilizado é baseado no preenchimento de uma região da tela delimitada por um polígono fechado. Ele é um algoritmo recursivo, que realiza os seguintes passos:

1. Verifica se o pixel atual possui a cor do polígono do limite de região.
2. Caso negativo, substitui a cor do pixel atual pela nova cor definida e chama o algoritmo recursivamente para seus quatro pixels vizinhos (N, S, L e O).
3. Caso afirmativo, termina a execução da função.

O pseudo-código para o Flood-fill é apresentado a seguir.

```
procedimento flood-fill (x, y, cor_nova, cor_limite)
inicio
  se pixel(x,y) = cor_limite então
    retorne;

  pixel(x,y) ← cor_nova;

  flood-fill(x+1, y, cor_nova, cor_limite);
  flood-fill(x-1, y, cor_nova, cor_limite);
  flood-fill(x, y+1, cor_nova, cor_limite);
  flood-fill(x, y-1, cor_nova, cor_limite);

fim_procedimento;
```

Este algoritmo terá que ser adaptado para o jogo de minas. Assim as mudanças serão:

- O limite de preenchimento são as casas não-vazias.
- A vizinhança de cada casa é de 8 pixels e não mais de 4 pixels.

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 ativar as 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, conforme visto no capítulo 3. Ele irá determinar se o jogo é do tipo 1, 2, se o jogador quer ver o ranking de tempos ou 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 “newGame”, que é responsável por rodar as rotinas de criação de mapa do arquivo “map.pas” e ajustar os valores de algumas variáveis.
- Passar o controle para o loop principal do jogo.

O código da rotina que inicia um novo jogo, após o jogador teclar enter no menu principal é descrito a seguir.

```
if i<3 then
begin
  gameType:=i;
  newGame;
  Color(15,0,0);
  cls2;
  Color(6,14,0);
  mountTable;
  gameControl;
end;
```

O código da rotina “newGame” é apresentado a seguir.

```
procedure newGame;
begin
  chooseGame(gameType);
  resetMap;
  createMap;
  flags:=mines;
  cursorX:=1;
  cursorY:=1;
  timeStart:=false;
end;
```

A rotina “chooseGame” irá ajustar os valores de algumas variáveis, de acordo com o tipo de jogo escolhido. As variáveis “cursorX” e “cursorY” determinam a posição corrente

da matriz. O flag “timeStart” indica se a contagem de tempo já começou. Isto permite que o tempo só comece a contar a partir do momento que o jogador abrir a primeira casa do jogo.

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 no controle do jogo. A função “keypressed” passa a tratar o evento de teclas somente quando uma tecla é pressionada. Ela é necessária, uma vez que não interrompe o loop a espera de uma tecla a ser pressionada, podendo atualizar constantemente o tempo de jogo.

O código dessa rotina é apresentado a seguir.

```
procedure gameControl;
var c : char;
    i,j : integer;
begin
  theEnd:=false;
  i:=1;
  j:=1;
  while theEnd=false do
  begin
    printLabels;
    drawCursor(i,j);
    if Keypressed then
    begin
      read(kbd,c);
      if (ord(c)=122) or (ord(c)=120) then
        checkTime;
      case ord(c) of
        28: begin { Right }
              i:=i+1; if i > width then i:=width;
            end;
        29: begin { Left }
              i:=i-1; if i < 1 then i:=1;
            end;
        31: begin { Down }
              j:=j+1; if j > height then j:=height;
            end;
        30: begin { Up }
              j:=j-1; if j < 1 then j:=1;
            end;
        122: begin { z - Open mine }
              openMine;
            end;
        120: begin { x - Check / Uncheck flags }
              changeFlag;
            end;
        99: begin { c - Open neighborhood }
              openNeighbors;
            end;
        27 : exit;
      end;
    end;
  readln;
end;
```

9

Contando o tempo de jogo

A maneira mais eficaz de se contar o tempo decorrente de um jogo é consultando o relógio do sistema. A partir dos MSX 2, o sistema passa a contar com um sistema de relógio. Assim, para se contar o tempo decorrido, basta anotar o horário no momento inicial de um evento, depois o horário no momento final e calcular a diferença dos dois horários.

O relógio do MSX trabalha apenas com as horas, os minutos e os segundos, diferente dos relógios do PC se são capazes de também contar os milésimos de segundos. Assim, nossa unidade de tempo é o segundo.

Há dois caminhos para se obter a hora corrente no MSX: a função REDCLK 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 abaixo¹:

GTIME (2CH)

Função: Leitura da hora.

Setup: Nenhum.

Retorno:

H = horas;

L = minutos;

D = segundos;

E = centésimos de segundo.

Apesar do sistema possuir um retorno para centésimos de segundo, o relógio do MSX não nos fornece esta informação. Portanto, a opção “E” deverá ser descartada.

Para utilizar um código em Assembly devemos usar o *inline* do Pascal. Os 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 primitivos para cada elemento da tabela: uma string para o nome do jogador e um inteiro para o tempo gasto. Assim, criaremos o type “rank”:

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

Para este jogo, criaremos uma tabela apenas com 20 posições do tipo “rank”, representando os dois jogos. Assim, cada jogo terá 10 posições de ranking.

```
var ranking : array[1..20] of rank;
```

O primeiro passo será limpar a tabela e assinalar com o valor -1, indicando que este espaço se encontra vazio. Então, temos:

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

Como essa tabela não recebe todos os elementos de uma só vez, nenhum algoritmo de ordenação será necessário. Desta forma, colocaremos os elementos um a um na tabela em ordem.

Quando o jogador sair do jogo, os resultados serão perdidos! assim, teremos que salvar os valores da tabela em disco. Primeiro, define-se um nome para o arquivo de saída, bem como uma variável de arquivo do tipo rank.

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

Em seguida, o programa para salvar o ranking é apresentado.

```

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 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;

```

Assim, se o arquivo de ranking existir, então carregue-o. Senão, limpe a tabela.

11

Corrigindo o problema do random

A geração de números aleatórios no Pascal apresentou alguns problemas, de forma que o resultado ficava um pouco “viciado”.

Para corrigir o problema da geração dos números aleatórios através da função “random”, deve-se aumentar o valor passado para essa função e depois trazer o resultado para o intervalo desejado.

```
var n, tam_faixa : integer;
...
randomize;
n := random(valor_grande) div tam_faixa;
```

Exemplo para o intervalo de 1 a 4:

```
n := random(4) + 1;
```

fica:

```
n := (random(100) div 4) + 1;
```

Devemos lembrar que a função *random(x)* gera um número aleatório entre 0 e $x-1$.

O “valor_grande” deverá ser um número múltiplo do tamanho estabelecido, senão haverá um desequilíbrio na distribuição dos resultados de *random(x)*. Exemplo:

```
n := (random(8) div 3) + 1;
```

Valores possíveis:

```
(0 div 3) + 1 = 1
(1 div 3) + 1 = 1
(2 div 3) + 1 = 1
(3 div 3) + 1 = 2
(4 div 3) + 1 = 2
(5 div 3) + 1 = 2
(6 div 3) + 1 = 3
(7 div 3) + 1 = 3
```

Nesse último exemplo, a probabilidade de dar 1 ou 2 é maior do que dar 3.

Qualquer outra função poderá ser usada para se obter aleatoriamente um número, desde que a frequência dos resultados seja a mesma para cada um dos valores da faixa estabelecida.

12

Créditos

O jogo MSX Minesweeper, bem como o manual foram elaborados por Marcelo Teixeira Silveira, Engenheiro de Sistemas e Computação, formado pela UERJ.

E-mail: flamar98@hotmail.com.

Homepage: <http://marmsx.msxall.com>

Agradecemos pela colaboração na construção desse jogo de SLotman, desenvolvedor de jogos e responsável pela página MSX Files.