MRDX 1.1: Novas Funcionalidades

Por Marcos Romero

http://romerogames.blogspot.com

Sumário:

26/04/2011

1- Introdução

Há 10 anos atrás eu concluí a implementação do MRDX (versão 1.0). O meu principal objetivo era facilitar o ensino da programação de jogos.

Mesmo sendo antigo, percebi que o propósito do MRDX continua válido até hoje, por isso resolvi ressuscitar o MRDX.

Criei a versão 1.1 para deixar registrado essa data de aniversário de 10 anos do MRDX.

Iniciarei em meu blog (romerogames.blogspot.com) uma série de postagens sobre técnicas de programação de jogos em 2D. Cada postagem será acompanhada de um exemplo prático feito com o MRDX para que as pessoas possam ver um código real aplicando a técnica que está sendo apresentada.

O objetivo é que os leitores entendam os conceitos apresentados e sejam capazes de implementar em sua linguagem/plataforma favorita.

2- Funções do MRDX 1.1

As novidades do MRDX 1.1 consistem em duas funções com melhorias e duas funções novas.

Mas, foi adicionado só isso? :)

Sim. A principal característica do MRDX pode ser definida em uma palavra: **simplicidade**.

A idéia por trás do MRDX é a de fornecer ao programador os elementos essenciais para que ele possa começar a programar jogos sem muitas complicações. Pensem no MRDX como o ABC da programação de jogos.

Veremos agora a descrição das novas funcionalidades do MRDX 1.1.

Protótipo: **int Carregar_Quadro(IMAGEM &imagem, int quadro, int cx, int cy,**
bool borda = false, int origemX = 0, int origemY = 0);

Comentários:

Esta função já existia no MRDX 1.0. É através dela que extraímos uma imagem de um bitmap que foi previamente carregado na memória e armazenamos em uma estrutura IMAGEM.

No MRDX 1.1 ela recebeu três novos parâmetros. Esses parâmetros já possuem valores padrões permitindo que a função possa ser chamada da mesma forma como era em sua versão anterior.

O parâmetro **borda** é booleano (true ou false) e serve para indicar se existe uma borda ou grade separando as imagens no bitmap. É muito comum a existência dessa grade nos arquivos com coleções de Sprites.

Os dois últimos parâmetros **origemX** e **origemY** servem para indicar um novo ponto de origem para ser levado em consideração ao extrair a imagem, ao invés de usar as coordenadas (0,0) do bitmap. Isso é útil quando a imagem desejada está posicionada de uma forma aleatória no bitmap.

Exemplo:

```
//quadro 3 da Imagem "jogador", célula (2,1) com bordas
Carregar_Quadro(jogador, 3, 2, 1, true);

//quadro 1 da Imagem "jogador", célula (0,0) começando na posição (400,400) sem bordas
Carregar_Quadro(jogador, 1, 0, 0, false, 400,400);
```

Protótipo: **int Desenhar_Imagem(IMAGEM &imagem,**
int espelhar = 0, int escalaX = 0, int escalaY = 0);

Comentários:

Esta função já existia no MRDX 1.0. Ela desenha uma variável do tipo IMAGEM na tela se baseando nas informações internas da estrutura IMAGEM.

No MRDX 1.1 ela recebeu três novos parâmetros. Esses parâmetros já possuem valores padrões permitindo que a função possa ser chamada da mesma forma como era em sua versão anterior.

O parâmetro **espelhar** indica se a imagem deve ser invertida na horizontal e/ou na vertical no momento do desenho. Este parâmetro pode receber os seguintes valores:

```
#define     MRDX_NAO_ESPELHAR          0
#define     MRDX_ESPELHAR_HOR          1
#define     MRDX_ESPELHAR_VERT         2
#define     MRDX_ESPELHAR_HOR_VERT     3
```

Nos jogos 2D com visão lateral os personagens são desenhados apenas em uma direção para economizar memória. A imagem do personagem olhando para o outro lado é gerada no momento do desenho através de uma inversão na horizontal. Isso pode ser visto na imagem abaixo do personagem Vega do Street Fighter 2. Reparem que em uma imagem a garra está na mão direita e na outra imagem a garra está na mão esquerda.

Os dois últimos parâmetros **escalaX** e **escalaY** servem para que a imagem seja desenhada com um tamanho diferente de seu tamanho original. Basta passar o valor da nova largura no parâmetro **escalaX** e o valor da nova altura no parâmetro **escalaY**.

Exemplo:

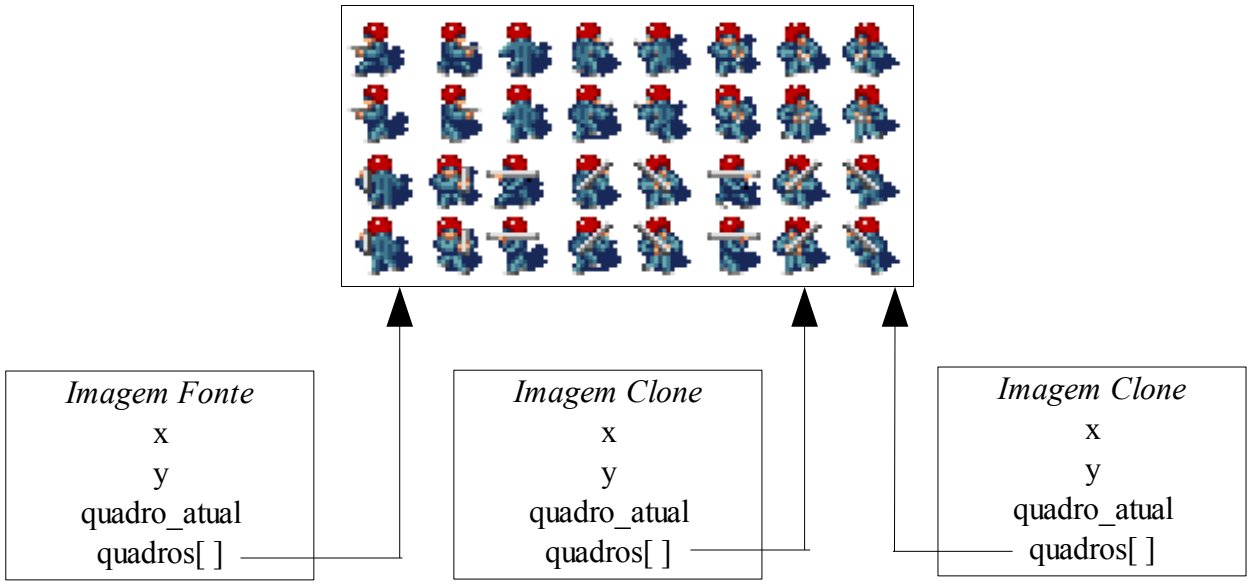
```
Desenhar_Imagem(jogador, MRDX_ESPELHAR_HOR);  //inverte na horizontal

Desenhar_Imagem(jogador, 0, 100, 100);     //Desenha com o novo tamanho de (100,100)
```

Protótipo: **int Clone_Imagem(IMAGEM &imagem_fonte, IMAGEM &imagem_destino);**

Comentários:

Esta é uma função nova do MRDX 1.1. Através dela é possível compartilhar os quadros de imagens que estão na memória do computador para diversas variáveis do tipo IMAGEM.

Para usar esta função é preciso criar e preparar uma variável IMAGEM com todos os quadros que ela irá utilizar. Essa será a IMAGEM FONTE. Após isso, use o método Clone_Imagem( ) para fazer cópias dessa IMAGEM FONTE. As variáveis IMAGEM geradas a partir deste método não precisam liberar memória através do método Destruir_Imagem( ).



Exemplo:

```
IMAGEM inimigo_fonte;
IMAGEM inimigo_clones[10];
//preparar inimigo_fonte (Criar, carregar quadro, etc...)
//Gerar 10 clones:
for( int i=0; i < 10; i++) {
        Clone_Imagem(inimigo_fonte, inimigo_clones[ i ]);
}
//Ao encerrar jogo liberar memória apenas do inimigo_fonte. Para os clones não precisa.
Destruir_Imagem( inimigo_fonte );
```

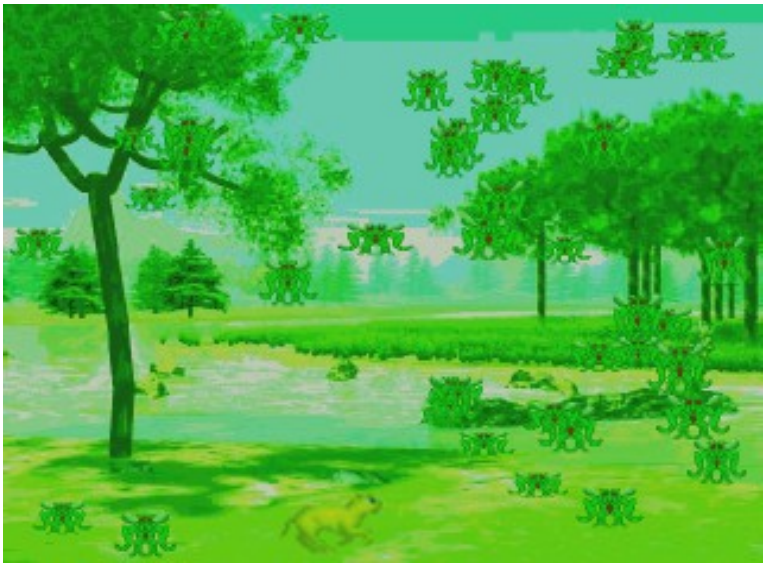
Protótipo: **int Transicao_Tela(int cor);**

Comentários:

Esta é uma função nova do MRDX 1.1. Ela altera lentamente todas as cores da tela para a cor que foi passada como parâmetro. Muito útil para mudança de fases ou telas do jogo.

As constantes de cores básicas do MRDX são PRETO, BRANCO, VERDE, AZUL, VERMELHO, AMARELO, CINZA e ROXO . No MRDX 1.1 essas cores ocupam as últimas posições da palheta de cores (248 a 255) para facilitar a manipulação das palhetas de cores das imagens usadas em um jogo do MRDX..

Na imagem abaixo está ocorrendo uma transição de tela usando a cor VERDE.



Exemplo:

```
//Sair do Jogo se a tecla Esc for pressionada
if( Testar_Tecla( DIK_ESCAPE ) ) {
        Transicao_Tela( VERDE ) ;
        Finalizar_Jogo( ) ;
        return ;
}
```

3- Exemplo de um Jogo

Vamos comentar agora a implementação do jogo de exemplo que acompanha a distribuição do MRDX 1.1. O código está no arquivo "jogo_mrdx1_1.cpp" na pasta "Exemplos\jogo_mrdx1_1".

3.1- **Definições de variáveis**

No início do código são definidas as variáveis do tipo IMAGEM que serão usadas no jogo, da seguinte forma:

```
IMAGEM imagem_fundo;
IMAGEM animal;
IMAGEM inseto;
IMAGEM insetos_clone[MAX_INSETOS];
```

A variável *imagem_fundo* é usada para representar o background. A variável *animal* representa o personagem controlado pelo jogador. A variável *inseto* contém as imagens e dados do personagem controlado pelo computador. Essa variável *inseto* servirá como **imagem fonte** para a matriz de variáveis *insetos_clone*[ ].

Cada clone do inseto possui suas próprias variáveis *x,y, quadro_atual* e *estado,* que são definidas na estrutura IMAGEM. Também foram definidos alguns dados extras que são armazenados na matriz dados[ ] da estrutura IMAGEM. As posições na matriz desses dados extras foram definidos nas seguintes constantes:

```
const int INSETO_DADOS_VEL_X = 0;
const int INSETO_DADOS_VEL_Y = 1;
const int INSETO_DADOS_LARGURA = 2;
const int INSETO_DADOS_ALTURA = 3;
```

Esses dados extras são usados para que cada clone do inseto possa ter velocidade e tamanho diferentes do *inseto fonte*.

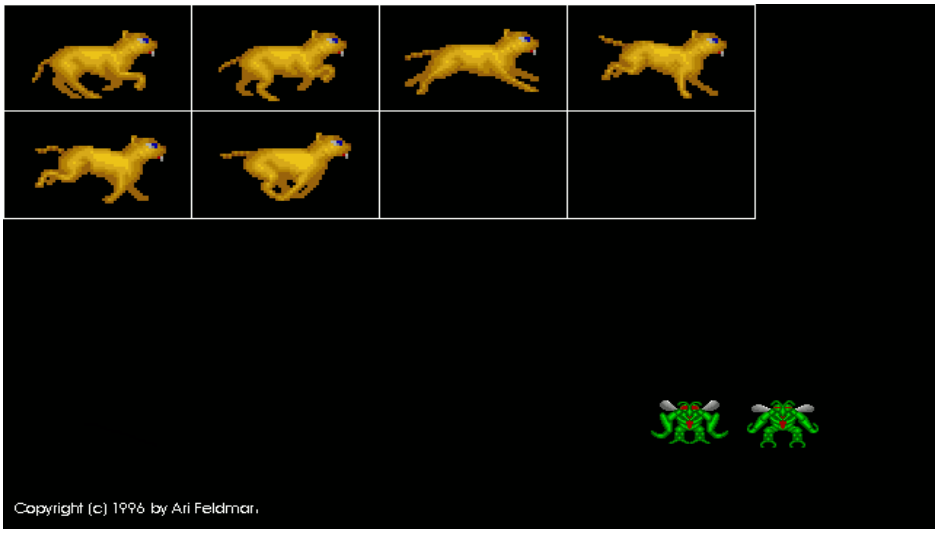
3.2 - **Jogo_Main( )**

A função Jogo_Main( ) é o ponto de entrada para um jogo feito com o MRDX. Ela é chamada constantemente durante a execução do programa. Neste jogo de exemplo ela apenas direciona o fluxo do programa para uma função de acordo com o estado atual do jogo como pode ser visto no código abaixo:

```
switch(estado_jogo) {
      case ESTADO_JOGO_INICIO:
                              Jogo_Inicio();
                              break;
      case ESTADO_JOGO_EXECUCAO:
                              Jogo_Execucao();
                              break;
      case ESTADO_JOGO_FIM:
                              Jogo_Fim();
                              break;
}
```

3.3 – **Jogo_Inicio( )**

Nesta função são inicializados os personagens do jogo com as imagens e dados iniciais. O bitmap “bichos.bmp” contém os quadros que serão usados pela IMAGEM *animal* e pela IMAGEM *inseto*. Os quadros do *animal* estão separados por bordas e os quadros do *inseto* estão no canto inferior direito do bitmap. Organizei o bitmap dessa forma para mostrar o uso das novas opções da função **Carregar_Quadro( )** no MRDX 1.1.



"bichos.bmp"

Depois que a IMAGEM *inseto* é inicializada são feitos vários clones através da função **Clone_Imagem( )** do MRDX 1.1. Para cada clone é definido aleatoriamente a velocidade e tamanho.

3.4 - **Jogo_Fim( )**

Nesta função é feita a liberação de memória utilizada pelas variáveis do tipo IMAGEM. Mas antes disso é chamada a função **Transicao_Tela( )** do MRDX 1.1, conforme pode ser visto abaixo:

```
Transicao_Tela(VERDE);
Destruir_Imagem(imagem_fundo);
Destruir_Imagem(animal);
Destruir_Imagem(inseto);
Finalizar_Jogo();
```

3.5 - **Jogo_Execucao( )**

Nesta função ocorre a lógica principal do jogo. São usadas as funções Iniciar_Relogio( ) e Esperar_Relogio ( ) para fixar o tempo gasto em um quadro do jogo em 40 milisegundos, resultando em 25 quadros por segundo. É feita a limpeza da tela no início e no final é exibida a tela que foi renderizada neste quadro. A imagem_fundo é desenhada e são chamadas as funções que irão processar a lógica do animal e dos insetos.

O código completo da função Jogo_Execucao( ) se encontra logo abaixo:

```
Iniciar_Relogio();
Limpar_Tela();
if(!Midi_Tocando(musica_mid)) {    //tocar música
      Tocar_Midi(musica_mid);
}
if(Testar_Tecla(DIK_ESCAPE)) {  //Sair do Jogo
      estado_jogo = ESTADO_JOGO_FIM;
      return;
}
Desenhar_Imagem(imagem_fundo);
Processar_Animal();
Processar_Insetos();
cont++;
Mostrar_Tela();
Esperar_Relogio(40);
```

3.6 - **Processar_Animal( )**

Esta função contém a lógica do *animal* que é controlado pelo jogador. É feita a atualização do quadro_atual que será desenhado. O *animal* pode ser movimentar para a esquerda e para direita, mas é feito um teste para evitar que ele saia da tela. A direção na qual o *animal* está olhando é baseada na tecla que o jogador pressionou. Para desenhar o *animal* olhando para a esquerda foi utilizado o parâmetro **espelhar** da função **Desenhar_Imagem( )** do MRDX 1.1.

3.7 - **Processar_Insetos( )**

Nesta função são processados todos os insetos que aparecem no jogo. É feito um laço que percorre a matriz *insetos_clone[ ]*. Para cada inseto é verificado se ele está ativo, é feita a atualização de sua posição atual de acordo com sua velocidade. Caso encoste na borda da tela ele muda de direção. Caso ocorra uma colisão com o *animal* o inseto passa para o estado INATIVO.

No final da função cada inseto é desenhado com seu próprio tamanho diferenciado devido ao uso dos parâmetros **escalaX** e **escalaY** da função **Desenhar_Imagem( )** do MRDX 1.1.

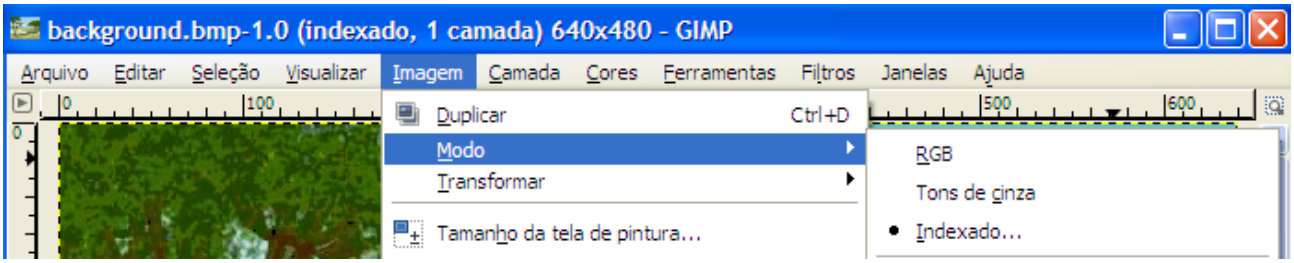
4- Extras

4.1 – **Imagens em modo indexado com palheta de 256 cores.**

As imagens usadas nos jogos do MRDX utilizam palheta de cores com 256 posições. Apenas para uma referência, os jogos do Super Nintendo também usam palhetas com 256 cores.

Todas as imagens precisam estar no formato BMP sem compressão e com a mesma palheta de cor, pois apenas uma palheta ficará ativa no momento da execução do jogo.

Na ferramenta GIMP (2.6.11) essa operação de conversão de imagens entre o modo RGB e Indexado está disponível no menu "Imagem -> Modo ->". Conforme imagem abaixo:

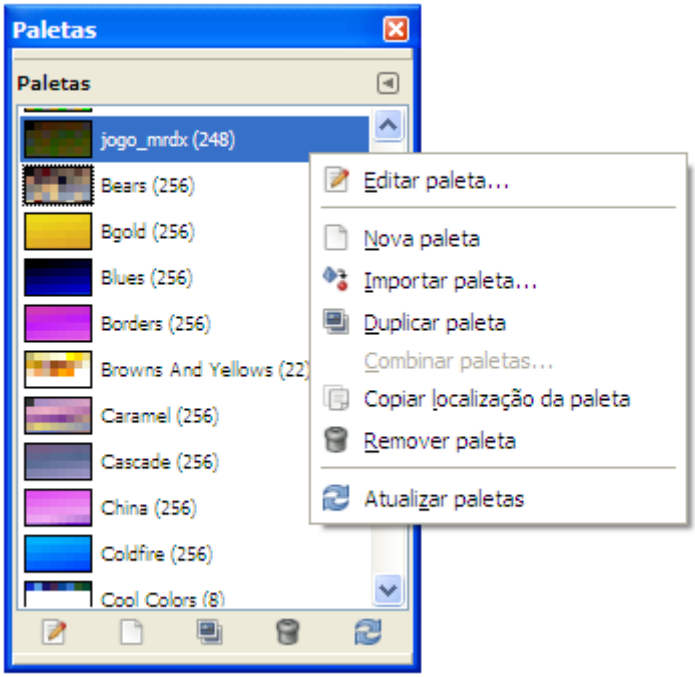


Ao criar palhetas para os jogos do MRDX é importante lembrar que a cor de índice 0 (zero) é a cor transparente que não será desenhada na tela do jogo, e que as 8 posições finais da palheta (248 a 255) são usadas pelo MRDX para as cores básicas.

Para editar as palhetas de cores no GIMP acesse o *Diálogo de Paletas,* que está no menu Janelas.

Tem várias opções acessíveis com o botão direito do mouse como mostra a figura ao lado.

A opção "*Combinar paletas...*" é muito útil mas ainda não foi implementada nessa versão do GIMP. Para contornar esse problema é possível ir direto no arquivo de texto que guarda os dados da palheta e fazer a combinação manualmente.



4.2 – **Tirando foto de jogos com DirectX usando a tecla PrintScreen.**

Por padrão a tecla *PrintScreen* não consegue copiar corretamente a imagem de um jogo que esteja sendo executado em modo de tela cheia com o DirectX. O resultado geralmente é uma tela toda escura ou com as cores trocadas. Vou mostrar agora o que precisa ser feito para que a tecla *PrintScreen* passe a funcionar corretamente nos jogos com DirectX.

Vá na opção "*Executar...*" que se encontra no menu *Iniciar* do Windows. Digite o comando "**regedit**" para abrir o Editor de Registro. Abra a chave "*HKEY_Local_Machine\Software\Microsoft\DirectDraw*". Clique com o botão direito do mouse e escolha "*Novo-> Valor DWORD*", coloque o nome **EnablePrintScreen** e insira o valor **1**, conforme figura abaixo.

Pronto, quando o Windows iniciar de novo ele habilitará o *PrintScreen* para DirectX. Para usar pressione *PrintScreen* durante a execução do jogo e depois cole em um editor de imagens.