Programando Jogos em Delphi

Animação, Lógica e Controle
versão orientada à objetos

Antônio Sérgio de S. Vieira

[1]Corrigida e Melhorada - 2011

sergiosvieira@hotmail.com

1. Introdução

Este texto foi escrito para pessoas que se interessam pela arte de programação de jogos, uma das mais completas, pois utiliza tanto a parte visual, quanto a sonora e interativa. Este trabalho não tem como objeto de estudo a criação do enredo do jogo e sim a parte lógica de seu desenvolvimento utilizando das facilidades do Delphi. Nesta nova versão deste tutorial, utilizou-se programação orientada à objetos.

É de suma importância que o leitor possua um bom nível em programação e algum conhecimento em ciências exatas, pois serão vistos assuntos específicos que utilizam lógica, matemática e física. [1]

No desenrolar da leitura serão abordadas questões relacionadas a manipulação de resoluções do vídeo, utilização de músicas e dos controles do jogo. Também será abordada a parte lógica do comportamento do personagem principal e de seus inimigos [2].

Neste primeiro exemplo, será criado um jogo de nave com personagem principal e tiros. Para isto, nesta nova versão do tutorial, resolvi utilizar programação orientada à objetos que ao meu ver facilita bastante o desenvolvimento de qualquer aplicação, principalmente jogos.

No segundo exemplo será explicado o desenvolvimento de um jogo estilo Mario Brothers, baseado em Blocos (*Tile Based Game*).[3]

[1]Neste primeiro tutorial não é preciso saber tanto.

[2]A ideia inicial era criar vários tutoriais sobre programação de jogos.

[3]Atualmente estou escrevendo um livro sobre programação de jogos, onde finalmente vou abordar este assunto.

2. Posicionando o Personagem do Jogo na Tela

A primeira coisa que se deve ter em mente é que o posicionamento do personagem no "cenário" (formulário do Delphi) é definido segundo um plano cartesiano invertido. Plano cartesiano para quem não lembra, é um local (plano) onde através de coordenadas (x, y) pode-se definir a posição de um ponto, e é através dele que definimos a posição de qualquer objeto em um jogo. Porém, diferente do plano cartesiano usual, os valores de y aumentam de cima para baixo. Observe a Figura 2:

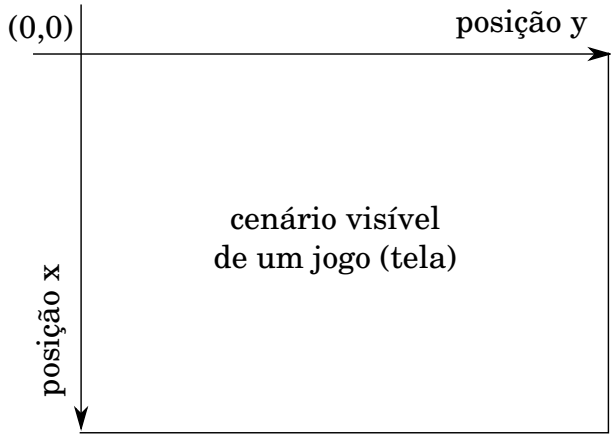

Figura 1. Cenário visível de um jogo segundo o plano cartesiano

Para descobrir em qual posição está o personagem ou qualquer objeto do jogo basta verificar as suas coordenadas x e y no plano cartesiano.

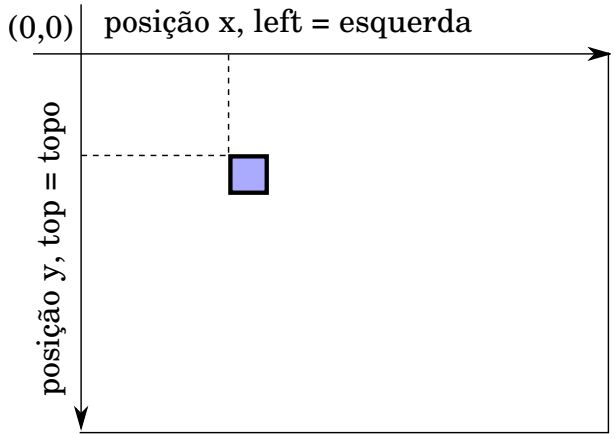

Figura 2. Posicionamento dos objetos no jogo

Estas coordenadas também são responsáveis pela movimentação dos objetos. Para movimentar, basta aumentar ou diminuir o valor de x para movimentar na horizontal ou de y para movimentar na vertical.

No código fonte 1, os valores das variáveis *posicao_x* e *posicao_y* são aumentados (**inc**)/diminuídos (**dec**) segundo o valor da variável *velocidade*. Isto quer dizer que quanto maior o valor da velocidade mais rápido será o movimento do personagem na tela.

Código Fonte 1. Exemplo de movimentação

```
program movimento;

const
  velocidade = 1;
  posicao_x, posicao_y: Integer;

begin
  posicao_x := 0; posicaoY := 0;
  inc(posicao_x, velocidade); // Movimento para a direita
  dec(posicao_x, velocidade); // Movimento para a esquerda
  inc(posicao_y, velocidade); // Movimento para baixo
  dec(posicao_y, velocidade); // Movimento para cima
end;
```

3. Entendendo o conceito de OffScreen

OffScreen é uma técnica utilizada para eliminar o "flick" (tremor) da animação, deixando os objetos do jogo com uma movimentação mais suave. Este técnica consiste em desenhar inicialmente todo o cenário (fundo, personagens etc) em uma tela virtual (em memória) e só em seguida desenhar o conteúdo da tela virtual no vídeo para exibição.

Para criar um *OffScreen* basta definir uma variável global do tipo *TBitmap* e depois de instanciá-la (coloque o código no *OnCreate* do formulário principal), defina valores para sua largura e altura, nunca esquecendo de destruí-la ao final do programa (*OnDestroy*).

Código Fonte 2. Exemplo de criação de OffScreen

```
(...)
var OFFScreen: TBitmap;
(...)

procedure TForm1.FormCreate(Sender: TObject);
begin
  OFFScreen:= TBitmap.create;
  OFFScreen.Width:= 320;
  OFFScreen.Height:= 240;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  OFFScreen.Free;
end;
```

4. Laço Principal do Jogo

O código principal do jogo (lógica, controle, movimentação e animação) deve estar dentro de um laço, para que seja possível realizar toda a dinâmica do jogo. Neste primeiro exemplo, utilizou-se um laço simples. Observe que não há nenhum tipo de controle sobre a movimentação do jogo. Isto quer dizer que dependendo da velocidade do computador que o jogo vá ser executado ele pode funcionar de forma mais rápida ou mais lenta. Veja abaixo um exemplo do código acima descrito:

Código Fonte 3. Exemplo de criação de OffScreen

```
procedure TForm1.Button1OnClick(sender:TObject);
begin
  while not Application.Terminated; // Em quanto jogo nao termina faca
    begin
      {CODIGO PRINCIPAL DO JOGO}
      Application.ProcessMessages; // utilizado para liberar processamento atual
    end;
end;
```

Neste exemplo, o "jogo" será iniciado ao clicar em um botão e o laço funcionará até que o aplicativo seja fechado. Dentro do laço deve-se colocar os comandos de manipulação do jogo e liberar o processamento através do comando *Application.ProcessMessages*.

5. Definindo Características dos Personagens do Jogo

Para facilitar a programação, é bom criar algumas classes. Elas são utilizadas para definir características e ações que um objeto do jogo pode realizar. Usar programação orientada à objetos facilita a programação, pois nós proporciona o reúso de código. No código 4 são criadas algumas variáveis e métodos importantes que definem como devem se comportar os objetos do jogo.

Código Fonte 4. Características que definem o comportamento de um personagem

```
1  Type
2    TSentido = (cima,baixo,esquerda,direita,parado);
3
4  Type
5    TNave = class
6    private
7      x: Integer; //Posicao horizontal
8      y: Integer; //Posicao vertical
9      velocidade: Integer;
10     sentido: TSentido; //Sentido do movimento
11     vivo: Boolean; //Controle de existencia
12     procedure ler_teclado();
13     procedure mover_personagem();
14   public
15     constructor create(x_, y_, velocidade_: Integer);
16     procedure mover();
17     procedure desenhar(offscreen: TBitmap);
18     procedure atirar();
19   end;
```

Na linha 2, foi criado um tipo *TSentido* que serve para informar qual o sentido atual do movimento do objeto. Já na linha 5, foi definida uma classe *TNave*, ela será responsável por representar o personagem de jogo. Nela, colocamos todas as características que queremos, assim como as ações que a nave deve realizar. Por exemplo, nas linhas 14, 15 e 16 são definidos procedimentos, eles servem para indicar quais ações o objeto pode realizar (mover-se, desenhar na tela e atirar). Outras características podem ser colocadas nesta classe. Mas para este primeiro tutorial somente estas bastam.

Caso não saiba a finalidade das palavras *private* e *public*, aqui cabe uma explicação. Elas definem permissões de acesso as variáveis (x, y, velocidade etc) para o programador (quem vai usar a classe TNave). Isto quer dizer que as variáveis ou métodos que forem privadas só podem ser acessadas dentro dos métodos da classe TNave. E as variáveis públicas podem ser acessadas durante o desenvolvimento do jogo, ou seja, fora dos métodos da classe TNave.[4]

6. Movimentando o Personagem

A movimentação do personagem pode ser dividida em dois métodos: no primeiro, as teclas pressionadas são identificadas para definir em que sentido o personagem deve se movimentar ou se ele deve atirar (*ler_teclado*). No segundo (*mover_personagem*), o sentido armazenada na variável *sentido* é usada para decidir como a posição (x, y) do personagem deve ser incrementada ou decrementada.

Código Fonte 5. Melhoramentos na movimentação do personagem

```
1  procedure TNave.ler_teclado();
2  begin
3    sentido := parado;
4    if (GetKeyState(VK_LEFT) < 0) then
5      sentido := esquerda
6    else if (GetKeyState(VK_RIGHT) < 0) then
7      sentido := direita;
8
9    if (GetKeyState(VK_UP) < 0) then
10     sentido := cima
11   else if (GetKeyState(VK_DOWN) < 0) then
12     sentido := baixo;
13
14   if (GetKeyState(VK_SPACE) < 0) then atirar();
15 end;
16
17 procedure TNave.mover_personagem();
18 begin
19   case sentido of
20     cima: dec(y, velocidade);
21     baixo: inc(y, velocidade);
22     esquerda: dec(x, velocidade);
23     direita: inc(x, velocidade);
24   end;
25 end;
```

Na linha 3, é definido que o personagem deve permanecer parado caso nenhuma tecla de movimentação seja pressionada. Quando uma das setas de movimentação do teclado for pressionada é definido o sentido do movimento do personagem. Isto é realizado nas linhas 4-12. Por último, na linha 14, quando pressiona-se a tecla espaço o método atirar de TNave é chamado.

A movimentação do personagem é realizada verificando o valor de *sentido*, como pode ser observados nas linhas 17-25, dependendo deste valor, os valores de x e y são aumentados ou diminuídos.

[4] Sei que existe muito mais coisas sobre isso, mas resolvi deixar simples

7. Desenhando no OffScreen

Para desenhar todo o cenário do jogos, como dito anteriormente, é necessário desenhar inicialmente na tela virtual (*OffScreen*). Isto é feito utilizando o método *desenhar* da classe TNave. O único parâmetro deste método é justamente a tela virtual.

Código Fonte 6. Desenhando na tela virtual

```
1 procedure TNave.desenhar(offscreen_: TBitmap);
2 var i, j: Integer;
3 begin
4   offscreen_.Canvas.FillRect(Rect(0, 0, 640, 480));
5   for i := 0 to 31 do
6     for j := 0 to 31 do
7       offscreen_.Canvas.Pixels[x + i, y + j] := clBlack;
8 end;
```

Na linha 4, a tela virtual é apagada. Neste exemplo foi utilizado uma tela com 640px de largura por 480px de altura. Os dois laços nas linhas 5-6 servem para desenhar um quadrado na tela (a nave :)). O desenho na tela virtual é feito através do comando na linha 7.

Para desenhar a tela virtual no formulário, basta usar o comando *Canvas.Draw(0, 0, OffScreen);*.

8. Atirando

Para fazer com que a nave atire é necessário criar uma nova classe. Ela vai representar o tiro no jogo. De forma semelhante a classe TNave, existem métodos para desenhar e movimentar o tiro.

Código Fonte 7. Classe TTiro

```
1   TTiro = class
2   private
3     x, y: Integer;
4     velocidade: Integer;
5     ativo: Boolean;
6   public
7     constructor create(velocidade_: Integer);
8     procedure mover();
9     procedure desenhar(offscreen_: TBitmap);
10  end;
```

Os dois principais métodos da class TTiro são *mover* e *desenhar*. No primeiro, a movimentação do tiro é realizada sem qualquer intervenção do usuário do jogo, ou seja, ele não precisa pressionar nenhum botão (além do espaço, claro) para fazer com que o tiro movimente-se pela tela. Para isto, pasta que o valor de y seja decrementado.

Código Fonte 8. Movimentando o tiro pela tela do jogo

```
1 procedure TTiro.mover;
2 begin
3   dec(y, velocidade);
4
```

```
5     if (y + 15 < 0) then
6     begin
7       ativo := false;
8     end;
9
10  end;
```

O mais importante do código 8, é o que está nas linhas 5-8. Quando a posição $y + 15 < 0$, o tiro será desativado. Isto quer dizer que ele já saiu da parte visível da tela, portanto não precisa mais se movimentar. A figura 3 pode melhorar o entendimento.

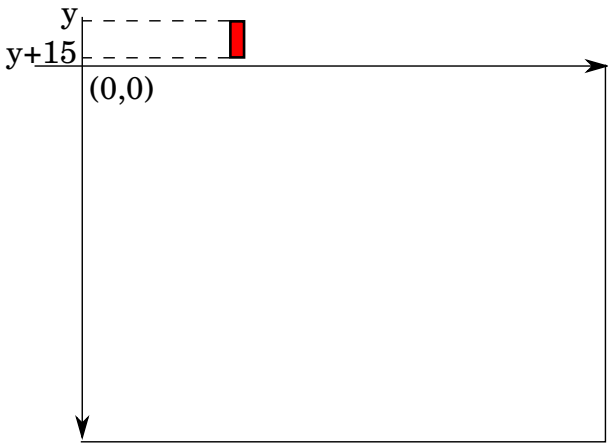

Figura 3. Posicionamento do tiro fora da parte visível do jogo

O método *desenhar* de TTiro é feito da mesma forma que o método *desenhar* de TNave.

9. Criando seu primeiro jogo no Delphi

Após termos visto alguns conceitos iniciais, podemos agora desenvolver o "jogo" de nave. Note que ainda não existirão inimigos e nem sprites. O jogo será apenas um quadrado movimentando e atirando pela tela. Sigua os passos descritos abaixo:

1. Crie um novo projeto no Delphi e na Unit1.pas, logo antes de *TForm1 = class(TForm)* e depois de `type` faça:
2. Crie um novo tipo para controlar o sentido do movimento da nave.

   ```
   TSentido = (cima, baixo, esquerda, direita, parado);
   ```

3. Crie a classe TTiro.

   ```
   TTiro = class
   private
     x, y: Integer;
     velocidade: Integer;
     ativo: Boolean;
   public
     constructor create(velocidade_: Integer);
     procedure mover();
     procedure desenhar(offscreen_: TBitmap);
   end;
   ```

4. Crie a class TNave

```
TNave = class
private
  x, y: Integer;
  sentido: TSentido;
  vivo: Boolean;
  velocidade: Integer;
  tiros: Array of TTiro;
  procedure ler_teclado();
  procedure mover_personagem();
  procedure mover_tiro();
public
  constructor create(x_, y_, velocidade_, quantidade_tiros: Integer;
            sentido_: TSentido);
  procedure mover();
  procedure desenhar(offscreen_: TBitmap);
  procedure atirar();
end;
```

5. Dentro da class *TForm1* crie o *OffScreen* e uma variável do tipo TNave.

```
{ Public declarations }
OffScreen: TBitmap;
spaceship: TNave;
```

6. Crie os método de TNave.

```
{TNave}

constructor TNave.create(x_, y_, velocidade_, quantidade_tiros: Integer;
                         sentido_: TSentido);
var i: Integer;
begin
  x := x_; y := y_;
  vivo := true;
  sentido := sentido_;
  velocidade := velocidade_;
  SetLength(tiros, quantidade_tiros);
  for i := 1 to quantidade_tiros do
    tiros[i] := TTiro.create(2);
end;

procedure TNave.mover();
begin
  ler_teclado();
  mover_personagem();
  mover_tiro();
end;

procedure TNave.ler_teclado();
begin
  sentido := parado;
  if (GetKeyState(VK_LEFT) < 0) then
    sentido := esquerda
  else if (GetKeyState(VK_RIGHT) < 0) then
    sentido := direita;
```

```
  if (GetKeyState(VK_UP) < 0) then
    sentido := cima
  else if (GetKeyState(VK_DOWN) < 0) then
    sentido := baixo;

  if (GetKeyState(VK_SPACE) < 0) then atirar();
end;

procedure TNave.mover_personagem();
begin
  case sentido of
    cima: dec(y, velocidade);
    baixo: inc(y, velocidade);
    esquerda: dec(x, velocidade);
    direita: inc(x, velocidade);
  end;
end;

procedure TNave.desenhar(offscreen_: TBitmap);
var i, j: Integer;
begin
  offscreen_.Canvas.FillRect(Rect(0, 0, 640, 480));
  for i := 0 to 31 do
    for j := 0 to 31 do
      offscreen_.Canvas.Pixels[x + i, y + j] := clBlack;
  for i := 1 to length(tiros) do
    tiros[i].desenhar(offscreen_);
end;

procedure TNave.atirar;
var i: Integer;
begin
  for i := 1 to length(tiros) do
    if (tiros[i].ativo = false) then
    begin
      tiros[i].x := x + 16;
      tiros[i].y := y;
      tiros[i].ativo := true;
      break;
    end
end;

procedure TNave.mover_tiro;
var i: Integer;
begin
  for i := 1 to length(tiros) do
    tiros[i].mover();
end;
```

7. Crie os método de TTiro.

```
{ TTiro }

constructor TTiro.create(velocidade_: Integer);
begin
  velocidade := velocidade_;
  ativo := false;
```

```
end;

procedure TTiro.desenhar(offscreen_: TBitmap);
var i, j: Integer;
begin
  for i:= 0 to 7 do
    for j := 0 to 15 do
      offscreen_.Canvas.Pixels[x + i, y + j] := clRed;
end;

procedure TTiro.mover;
begin
  dec(y, velocidade);

  if (y + 15 < 0) then
  begin
    ativo := false;
  end;

end;
```

8. Por último só falta definir os métodos de *TForm*. Em *Events* do *Object Inspector* clique duas vezes em *OnCreate* e coloque.

```
{TForm1}

procedure TForm1.FormCreate(Sender: TObject);
begin
  spaceship := TNave.create(320, 240, 1, 3, parado);
  OffScreen := TBitmap.Create;
  OffScreen.Width := 640;
  OffScreen.Height := 480;
end;
```

9. Em *Events* do *Object Inspector* clique duas vezes em *OnDestroy* e coloque.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  OffScreen.Free;
end;
```

10. Em *Events* do *Object Inspector* clique duas vezes em *OnActivate* e coloque.

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  while not application.Terminated do
  begin
    spaceship.desenhar(OffScreen);
    spaceship.mover();
    Canvas.Draw(0, 0, OffScreen);
    application.ProcessMessages;
  end;
end;
```

11. tecle F9 e TCHARAMMMM!!! Seu primeiro jogo feito em Delphi.