```
unit AutoPeliForm;
(*
  Esimerkki
    1) Liikkuvan auton käytöstä
    2) Olion dynaamisesta luomisesta
    3) Olion lähettämän omatekoisen viestin vastaanottamisesta

  Tarvitsee toimiakseen
    AutoPeliForm.pas - tämä tiedosto
    sprite.pas       - kantalauokka koko liikuttelulle
    liikkuva.pas     - TSpritestä peritty liikkuva olio
    auto.pas         - TLiikkuvast peritty auto
    traffic.pas      - liikennevalot
    rusinaauto.pas   - auto joka syttyy palamaan ja kutistuu rusinaksi
    driversbrain.pas - autoon äly, joka osaa reagoida liikennevaloihin

  Vesa Lappalainen 14.11.1999

  08.09.2001/vl
  + VCL/CLX -käännös (määrittele vakio CLX)
*)
interface

uses
  SysUtils, Classes,
{$ifdef CLX}
  QGraphics, QControls, QForms, QDialogs, QExtCtrls,QStdCtrls,
{$else}
  Windows, Messages, Graphics, Controls, Forms, Dialogs, ExtCtrls,StdCtrls,
{$endif}
  Auto,sprite, traffic, DriversBrain, rusinaauto;

type
  TFormAutoPeli = class(TForm)
    PanelOptions: TPanel;
    EditAutoja: TEdit;
    LabelAutoja: TLabel;
    LabelKA: TLabel;
    EditKA: TEdit;
    LabelLV: TLabel;
    EditLV: TEdit;
    ButtonUusi: TButton;
    LabelPaloPros: TLabel;
    EditPaloPros: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure AutoMove(sender:TAuto);
    procedure KAutoChangeDir(sender: TObject; dir: Integer);
    procedure EditAutojaEnter(Sender: TObject);
    procedure ButtonUusiClick(Sender: TObject);
  private
    Auto : TAuto;
    space : TSpriteSpace;
  public
    procedure Uusi;
  end;

var
  FormAutoPeli: TFormAutoPeli;

implementation
uses vclclx;

{$R *.xfm}

procedure TFormAutoPeli.FormCreate(Sender: TObject);
begin
  Randomize();
  uusi;
  FormCaption(self);
```



```
end;

procedure TFormAutoPeli.AutoMove(sender:TAuto);
begin
//
end;

procedure TFormAutoPeli.KAutoChangeDir(sender: TObject; dir: Integer);
begin
//
end;

procedure TFormAutoPeli.EditAutojaEnter(Sender: TObject);
begin
  if not ( sender is TEdit ) then exit;
  TEdit(Sender).SelectAll;
end;

function Num(e:TEdit;m:integer=40):integer;
begin
  Result := StrToIntDef(e.text,1);
  if ( Result > m ) then Result := m;
end;

procedure TFormAutoPeli.Uusi;
var i:integer;  tl : TTrafficLights; n:integer;
begin
  Randomize();
  space := TSpriteSpace.Create(self);
  space.Parent := self;
  space.Align := alClient;
  n := Num(EditAutoja,300)+Num(EditKa,300);
  for i:=0 to n-1 do begin
    Auto := TRusinaksiMuuttuvaAuto.Create(space);
    TDriversBrain.Create(Auto);
    Auto.x := Random(Space.ClientWidth-50);
    Auto.y := Random(Space.ClientHeight-20);
  //  Auto.Parent := self;
    if ( Random(n) > Num(EditKA) ) then
      Auto.BitName := 'hauto.bmp'
    else
      Auto.BitName := 'kauto.bmp';
    Auto.Speed := Random(20)+1;
    Auto.RemoveOnHit := Random(100) < Num(EditPaloPros,100); // n% todennäköisyys olla sytty
    if ( Random(100) > 80 ) then // 20 % todennäköisyydellä "vinoon" liikkuva
    Auto.Direction := Random(360);
//    Auto.OnAutoMove := AutoMove;
//    Auto.OnChangeDir := KAutoChangeDir;
  end;

  for i:=0 to Num(EditLV,40)-1 do begin
    tl := TTrafficLights.Create(space);
    tl.x := Random(Space.ClientWidth-50);
    tl.y := Random(Space.ClientHeight-20);
    tl.Interval := 40 + Random(100);
  end;

  Space.Interval := 100;
  Space.Running := true;
end;

procedure TFormAutoPeli.ButtonUusiClick(Sender: TObject);
begin
  Space.Running := false;
  Space.Free;
  Uusi;
end;

end.
```

```
{------------------------------------------------------------------------------}
{
   Unit Name: sprite
   Purpose  : The basic class for sprites
   Author   : Vesa Lappalainen
   Date     : 03.11.1999
   Changed  : 5.11.1999
              - Avaruuteen Spriteille oma lista, jossa on kaikki avaruudessa
                olevat spritet
              - Törmäysten tunnistaminen ja reagointi
              - lisätty vielä törmäysten takia oma lista : FDead,
                johon siirreään ne Spritet, jotka ovat
                TSpriteSpace.UnJoin -metodilla ilmoittautuneet
                kuolevansa.  Lista tyhjennetään kun jokainen
                Sprite on saanut mahdollisuuden reagoida
                toisen kanssa.
              - vastaavasti jokainen sprite pitää liitää Sprite-listaan
                Join-metodilla (TSprite.Create-tekee tämän).
              14.11.1999
              - äly (TBrain).  Jokaisella TSpritella voi olla vain yksi
                äly ja kukin äly voi ohjata vain yhtä spriteä.
                Sprite tulee toimeen ilmankin älyä
                Äly saa ensin tarkistaa törmäyksen, jos ei tarkista,
                niin sitten itse sprite saa tehdä homman.
                Samoin React menee ensin älylle, jos ei käsitellä, niin
                Sprite saa käsitellä
   ToDo     : - kannattaisi tehdä mielluumin ensin suorakaide ja piste
                tyypit ja näille operaatiot.  Vähenisi nuo joku.x tutkiskelut.
              - muunnosmatriisin voisi ujuttaa avaruuteen, jolloin mukaan
                saisi vaikka kolmiulotteisuuden

Muutettu (Jarmo Louet, 20.11.1999)
 - Lisäsin eventit OnTimerFirst, OnTimerAfterJobs, OnTimerAfterHits ja
   OnTimerLast. Kaikki tapahtuvat DoTimer-silmukan aikana.
     1) Näistä OnTimerFirst tapahtuu aivan ensimmäiseksi, ennen mitään muuta.
     2) OnTimerAfterJobs tapahtuu nimensä mukaan heti DoJobeja kutsuvan
        silmukan jälkeen (mutta ennen kuin poistutaan DoTimeristä, jos ei
        tehty yhtään jobia).
     3) Vastaavasti OnTimerAfterHits tapahtuu IsHit ja React silmukoiden
        jälkeen, mutta ennen kuin kuolleet on poistettu.
     4) OnTimerLast tapahtuu aivan DoTimerin lopuksi.
 - Huvitti tehdä näistä eventtejä, jotta käyttäjä voi itse määritellä mitä
   niiden aikana tehdään ilman, että tarvitsee aina ruveta perimään ja
   overrideamaan. Toisaalta olisi voinut olla parempi käyttää nimenomaan
   overridea. Tiedä häntä... On tässä nyt ainakin se vaara olemassa, että
   onnistutaan katkaisemaan DoTimerin suoritus kokonaan ennen aikojaan.

Muutettu (Vesa Lappalainen, 19.10.2000)
- lisätty procedure DrawBackGround(bitmap: TBitmap); virtual;
  jotta jokainen voi piirtää oman taustakuvan ennen spritejen piirtoa.

Muutettu (Vesa Lappalainen, 08.09.2001)
- VCL/CLX -käännös

Muutettu (Vesa Lappalainen, 08.12.2002)
- lisätty procedure DrawForeGround(bitmap: TBitmap); virtual;
  jotta jokainen voi piirtää oman edustakuvan spritejen piirton jälkeen.

}
{------------------------------------------------------------------------------}

unit sprite;
interface

uses Classes,
{$ifdef CLX}
  QGraphics,QExtCtrls
{$else}
  Graphics,ExtCtrls
```



```
{$endif}
  ;

type
  TSpriteSpace = class;
  TBrain = class;
  TOnTimerFirst = procedure(sender:TSpriteSpace) of object;
  TOnTimerAfterJobs = procedure(sender:TSpriteSpace) of object;
  TOnTimerAfterHits = procedure(sender:TSpriteSpace) of object;
  TOnTimerLast = procedure(sender:TSpriteSpace) of object;

  TSprite  = class(TComponent)
  private
    Fx: double;
    Fy: double;
    Fw: double;
    Fh: double;
    FParent : TSpriteSpace;
    FBrain: TBrain;

    procedure Setx(const Value: double);
    procedure Sety(const Value: double);
    procedure Seth(const Value: double);
    procedure Setw(const Value: double);
    function GetParent: TSpriteSpace;
    function GetIx: integer;
    function GetIy: integer;
    procedure SetBrain(const Value: TBrain);
    function TakeBrain : TBrain; virtual;
    function GetIh: integer;
    function GetIw: integer;
  public
    constructor Create(AOwner:TComponent); override;
    destructor Destroy; override;
    function DoJob : boolean; virtual;
    function IsHit(sprite:TSprite):boolean; virtual;
    function React(sprite:TSprite):boolean; virtual;
    function ReactTo(sprite:TSprite):boolean; virtual;
    function PaintTo(canvas:TCanvas):boolean; virtual;
    property Parent : TSpriteSpace read GetParent;
    property Brain : TBrain read FBrain write SetBrain;
  published
    property x:double read Fx write Setx;
    property y:double read Fy write Sety;
    property h:double read Fh write Seth;
    property w:double read Fw write Setw;
    property ix : integer read GetIx;
    property iy : integer read GetIy;
    property iw : integer read GetIw;
    property ih : integer read GetIh;
  end;

  TBrain = class
  private
    FOwner: TSprite;
    procedure SetOwner(const Value: TSprite);
  public
    constructor Create(AOwner:TSprite); virtual;
    destructor Destroy; override;

    function DoJob : boolean; virtual;
    function IsHit(sprite:TSprite):boolean; virtual;
    function React(sprite:TSprite):boolean; virtual;
    function ReactTo(sprite:TSprite):boolean; virtual;

    property Owner : TSprite read FOwner write SetOwner;
  end;

  TSpriteSpace = class(TPanel)
```

```
  private
    bitmap : TBitmap;
    PaintBox : TPaintBox;
    timer : TTimer;
    FSprites : TList;
    FKill : Tlist;
    FRemove : Tlist;
    FOnTimerFirst : TOnTimerFirst;
    FOnTimerAfterJobs : TOnTimerAfterJobs;
    FOnTimerAfterHits : TOnTimerAfterHits;
    FOnTimerLast : TOnTimerLast;
    procedure SetInterval(const Value: integer);
    function GetInterval: integer;
    function GetRunning: boolean;
    procedure SetRunning(const Value: boolean);
    procedure DoTimer(sender: TObject);
  protected
    procedure Display;                      virtual;
    procedure SpacePaint(sender: TObject);  virtual;
    procedure SpaceResize(sender: TObject); virtual;
    procedure ReDraw; virtual;
    procedure DrawBackGround(bitmap: TBitmap); virtual;
    procedure DrawForeGround(bitmap: TBitmap); virtual;
  public
    constructor Create(AOwner:TComponent); override;
    destructor Destroy; override;
    property Sprites : TList read FSprites;
    procedure Join(s:TSprite);              virtual;
    procedure UnJoin(s:TSprite; LicenceToKill:boolean); virtual;
  published
    property Interval : integer read GetInterval write SetInterval;
    property Running : boolean read GetRunning write SetRunning;
    property OnTimerFirst:TOnTimerFirst read FOnTimerFirst write FOnTimerFirst;
    property OnTimerAfterJobs:TOnTimerAfterJobs read FOnTimerAfterJobs write FOnTimerAfterJo
    property OnTimerAfterHits:TOnTimerAfterHits read FOnTimerAfterHits write FOnTimerAfterHi
    property OnTimerLast:TOnTimerLast read FOnTimerLast write FOnTimerLast;
  end;


implementation

{ TSprite }

constructor TSprite.Create(AOwner: TComponent);
begin
  inherited;
  x := 0;
  y := 0;
  h := 20;
  w := 20;
  if ( AOwner is TSpriteSpace ) then begin
    FParent := TSpriteSpace(AOwner);
    Parent.Join(self);
  end;
end;

destructor TSprite.Destroy;
begin
  if ( Assigned(Brain) ) then Brain.Free;
  inherited;
end;

function TSprite.DoJob: boolean;
begin
//  if ( Brain <> nil ) and ( Brain.DoJob(sprite) ) then begin Result := true; exit; end;
//  Brain.DoJob(sprite) sanoo "too many actual parameters"
  if ( Brain <> nil ) and ( Brain.DoJob ) then begin Result := true; exit; end;
  Result := false;
end;
```



```
function TSprite.GetIh: integer;
begin
  Result := Round(h);
end;

function TSprite.GetIw: integer;
begin
  Result := Round(w);
end;

function TSprite.GetIx: integer;
begin
  Result := Round(x);
end;

function TSprite.GetIy: integer;
begin
  Result := Round(y);
end;

function TSprite.GetParent: TSpriteSpace;
begin
  Result := FParent;
end;

function TSprite.IsHit(sprite: TSprite): boolean;
begin
  Result := false;
  if ( Brain <> nil ) and ( Brain.IsHit(sprite) ) then begin Result := true; exit; end;
  if ( x + w < sprite.x ) then exit;
  if ( x > sprite.x + sprite.w ) then exit;
  if ( y + h < sprite.y ) then exit;
  if ( y > sprite.y + sprite.h ) then exit;
  Result := true;
end;

function TSprite.PaintTo(canvas: TCanvas): boolean;
begin
  Result := false;
end;

function TSprite.React(sprite: TSprite): boolean;
begin
  if ( Brain <> nil ) and ( Brain.React(sprite) ) then begin Result := true; exit; end;
  Result := sprite.ReactTo(self);
end;

function TSprite.ReactTo(sprite: TSprite): boolean;
begin
  if ( Brain <> nil ) and ( Brain.ReactTo(sprite) ) then begin Result := true; exit; end; //
  Result := false;
end;

procedure TSprite.SetBrain(const Value: TBrain);
begin
  if ( Brain = value ) then exit;
  if ( Brain <> nil ) then Brain.Free;  // Del old brains
  FBrain := Value;
  if ( Brain = nil ) then exit;
  Brain.Owner := self;
end;

procedure TSprite.Seth(const Value: double);
begin
  Fh := Value;
end;

procedure TSprite.Setw(const Value: double);
```

```
begin
  Fw := Value;
end;

procedure TSprite.Setx(const Value: double);
begin
  Fx := Value;
end;

procedure TSprite.Sety(const Value: double);
begin
  Fy := Value;
end;

function TSprite.TakeBrain: TBrain;
begin
  Result := Brain;
  FBrain := nil;
end;

{ TBrain }

constructor TBrain.Create(AOwner: TSprite);
begin
  inherited Create;
  Owner := AOwner;
end;

destructor TBrain.Destroy;
begin
  if ( Owner <> nil ) then begin
    Owner.TakeBrain;
    FOwner := nil;
  end;
  inherited;
end;

function TBrain.DoJob: boolean;
begin
  Result := false;
end;

function TBrain.IsHit(sprite: TSprite): boolean;
begin
  Result := false;
end;

function TBrain.React(sprite: TSprite): boolean;
begin
  Result := false;
end;

function TBrain.ReactTo(sprite: TSprite): boolean;
begin
  Result := false;
end;

procedure TBrain.SetOwner(const Value: TSprite);
begin
  if ( Owner = Value ) then exit;
  if ( FOwner <> nil ) then FOwner.TakeBrain;  // Old owner looses these brains
  FOwner := Value;
  if ( Owner = nil ) then exit;
  Owner.Brain := self;  // Owner take's care of deleting old brains
end;

{ TSpriteSpace }

constructor TSpriteSpace.Create(AOwner: TComponent);
```



```
begin
  Timer := TTimer.Create(self);
  inherited;
  bitmap := TBitmap.Create;
  PaintBox := TPaintBox.Create(self);
  FSprites := TList.Create;
  FKill := TList.Create;
  FRemove := TList.Create;
  OnResize := SpaceResize;
  PaintBox.Parent := self;
  PaintBox.OnPaint := SpacePaint;
  Timer.OnTimer := DoTimer;
  Interval := 50;
end;

destructor TSpriteSpace.Destroy;
begin
  Sprites.Free;
  FKill.Free;
  FRemove.Free;
  bitmap.Free;
  PaintBox.Free;
  Timer.Free;
  inherited;

end;

procedure TSpriteSpace.SpaceResize(sender:TObject);
begin
  bitmap.Width := Width;
  bitmap.Height := Height;
  Paintbox.Width := Width;
  Paintbox.Height := Height;
  ReDraw;
end;

procedure TSpriteSpace.Display;
begin
  PaintBox.Canvas.Draw(0,0,bitmap);
end;

procedure TSpriteSpace.DrawBackGround(bitmap:TBitmap);
begin
  bitmap.Canvas.Rectangle(0,0,bitmap.Width,bitmap.Height);
end;

procedure TSpriteSpace.DrawForeGround(bitmap:TBitmap);
begin
//
end;


procedure TSpriteSpace.ReDraw;
var i:integer; s:TSprite;
begin
  bitmap.Canvas.Brush.Color := Color;
  DrawBackGround(bitmap);
  for i:=0 to Sprites.Count-1 do begin
    s := Sprites[i];
    s.PaintTo(bitmap.canvas);
  end;
  DrawForeGround(bitmap);
  Display;
end;

procedure TSpriteSpace.DoTimer(sender:TObject);
var n,i,j:integer; s,s1:TSprite;
begin
  if Assigned(FOnTimerFirst) then FOnTimerFirst(self);
```

```
  n := 0;
  for i:=0 to Sprites.Count-1 do begin
    s := Sprites[i];
    if s.DoJob then inc(n);
  end;
  if Assigned(FOnTimerAfterJobs) then FOnTimerAfterJobs(self);
  if ( n = 0 ) then exit;
  for i:=0 to Sprites.Count-1 do begin
    s := Sprites[i];
    for j:=i+1 to Sprites.Count-1 do begin
      s1 := Sprites[j];
      if s.IsHit(s1) then s.React(s1);
    end;
  end;
  if Assigned(FOnTimerAfterHits) then FOnTimerAfterHits(self);
  for i:=0 to FRemove.Count-1 do begin
    Sprites.Remove(FRemove[i]);
  end;
  for i:=0 to FKill.Count-1 do begin
    s := FKill[i]; s.Free;
  end;
  FRemove.Clear;
  FKill.Clear;
  if Assigned(FOnTimerLast) then FOnTimerLast(self);

  ReDraw;
end;

procedure TSpriteSpace.SpacePaint(sender:TObject);
begin
  Display;
end;

procedure TSpriteSpace.Setinterval(const Value: integer);
begin
  Timer.Interval := Value;
end;

function TSpriteSpace.GetInterval: integer;
begin
  Result := Timer.Interval;
end;

function TSpriteSpace.GetRunning: boolean;
begin
  Result := Timer.Enabled;
end;

procedure TSpriteSpace.SetRunning(const Value: boolean);
begin
  Timer.Enabled := Value;
end;

procedure TSpriteSpace.Join(s: TSprite);
begin
  Sprites.Add(s);
end;

procedure TSpriteSpace.UnJoin(s: TSprite; LicenceToKill:boolean);
begin
  if LicenceToKill and ( FKill.IndexOf(s) < 0 ) then
    FKill.Add(s);
  FRemove.Add(s);
end;


end.
```



```
unit moving;
(*
  Esimerkki
   1) TImageListin käytöstä
   2) TPaintBoxin perimisestä
   3) TBitMapin käytöstä (mm. peilaaminen y-akselin suhteen)
   4) Viestin lähettämisestä (Event) omistajalle
       - auto lähettää aina liikahtaessaan viestin
       - auto lähettään viestin kun se kääntyy

   Vesa Lappalainen 23.10.1997

   Changed  : 5.11.1999/vl
    + peritty TSpritestä
   Changed  : 14.11.1999/vl
    + korjattu React-metodia
    + korjattu PaintTo-metodia siten, etta suunta päätetään suunnasta, ei dx:stä
    + lisätty MovingReact perittäväksi paremmin
   Changed  : 08.09.2001/vl
    + VCL/CLX-käännös
    + entinen TAuto muutettu TMovingSprite ja TAuto peritään TMovingSprite
    + direction changed to double and the unis is now degrees in math coordinate
*)

interface

uses
  SysUtils, Classes, Types,
{$ifdef CLX}
  QGraphics, QControls, QForms, QDialogs, QExtCtrls,
  QImgList,
{$else}
  Windows, Messages, Graphics, Controls, Forms, Dialogs,  ExtCtrls,
{$endif}
  sprite;

type
  TMovingSprite = class;

  TOnSpriteMove  = procedure (sender:TMovingSprite) of object;
  TOnChangeDir = procedure (sender:TObject;dir:integer) of object;

  TMovingSprite = class(TSprite)
  private
    FSpeed : double;
    FDirection : integer; // allways -180 - 180
    dx,dy   : double;
    FOnMove : TOnSpriteMove;
    FOnChangeDir : TOnChangeDir;
    FStayOnParentArea: boolean;
    procedure CountDirection;
  protected
    FBitName : String;
    FImageList : TImageList;
    procedure CountDXDY; virtual;
  public
    constructor Create(AOwner:TComponent); override;
    destructor  Destroy; override;
    procedure SetBitName(s:string); virtual;
    procedure SetSpeed(d:double); virtual;
    procedure SetDirection(i:integer); virtual;
    function DoJob:boolean; override;
    function PaintTo(canvas:TCanvas):boolean; override;
    function React(sprite:TSprite):boolean; override;
    function MovingReact(sprite:TSprite):boolean; virtual;
  published
    property BitName : string read FBitName write SetBitName;
    property Speed : double read FSpeed write SetSpeed;
    property Direction : integer read FDirection write SetDirection default 0;
    property OnMove : TOnSpriteMove read FOnMove write FOnMove;
    property OnChangeDir : TOnChangeDir read FOnChangeDir write FOnChangeDir;
    property StayOnParentArea : boolean read FStayOnParentArea write FStayOnParentArea default true;
  end;


implementation

uses Math;

constructor TMovingSprite.Create(AOwner:TComponent);
begin
```

```
  inherited;
  FImageList := TImageList.Create(self);
  FSpeed := 0;
  FDirection := 0;
  CountDXDY;
  FOnMove := NIL;
  FOnChangeDir := NIL;
  FStayOnParentArea := true;
end;

destructor TMovingSprite.Destroy;
begin
  FImageList.Free;
  inherited;
end;


procedure TMovingSprite.CountDXDY();
var rad : double;
begin
  rad := Direction * Pi / 180;
  dx := cos(rad);
  dy := sin(rad);
end;

procedure TMovingSprite.CountDirection();
var rad : double;
begin
  if ( dx = 0 ) then begin
    if ( dy < 0 ) then Direction := -90 else Direction := 90;
    exit;
  end;
  rad := ArcTan2(dy,dx) * 180 / Pi;
  Direction := Round(rad);
end;


procedure TMovingSprite.SetDirection(i:integer);
var change : boolean;
begin
  change := Direction <> i;
  FDirection := i;
  CountDXDY;
  if ( change ) and Assigned(FOnChangeDir) then FOnChangeDir(self,i);
end;


procedure TMovingSprite.SetSpeed(d:double);
begin
  FSpeed := d;
end;

procedure TMovingSprite.SetBitName(s:string);
// Kun bittikartan tiedoston nimi vaihdetaan, pyyhitään vanhat kuvat pois ja
// luodaan uudet kuvat.  Vasemmalle menevä kuva saadaan peilaamalla oikealle
// menevät kuva y-akselin suhteen.
var rect,rect2 : TRect;
    lbmp,rbmp :  TBitmap; // Vasemmalle ja oikealle menevät kuvat
begin
  FImageList.Clear;                             // Pyyhitään vanhat kuvat pois
  if ( s = '' ) then begin {Paint;} exit; end;     // Jos ei nimeä, ei kuvia
  rbmp := TBitmap.Create;                       // Luodaan tilap. bittikartat
  lbmp := TBitmap.Create;                       // Oik. ja avsemmalle

  FBitName := s;                                // Muistetaan kuvan nimi
  rbmp.LoadFromFile(s);                         // Luetaan Oik. kuva tied.

  lbmp.Assign(rbmp);                            // Tehdään vas. identtinen
  rect.Left   := 0;                             // Apusuorakaiteet kop.
  rect.Top    := 0;
  rect.Right  := rbmp.Width;
  rect.Bottom := rbmp.Height;
  rect2       := rect;                          // rect2 on rectin peilikuva
  rect2.Right := 0;
  rect2.Left  := rect.Right;
  lbmp.Canvas.StretchDraw(rect2,rbmp);       // lbmp on rbmp:n peilikuva
//  lbmp.Canvas.CopyRect(rect2,rbmp.Canvas, rect); // Ei toimi CLX:ssä

  w  := rbmp.Width;                          // Oma koko bitmapin kokoiseksi
  h := rbmp.Height;
  FImageList.Width  := rbmp.Width;                   // Samoin kuvalistan
```



```
  FImageList.Height := rbmp.Height;
  FImageList.AddMasked(rbmp,rbmp.TransparentColor); // Kuvalistan paik. 0 rbmp
  FImageList.AddMasked(lbmp,lbmp.TransparentColor); //                   1 lbmp
 // Paint;
  lbmp.Free;                                        // Poistetaan tilap. kuvat
  rbmp.Free;
end;

function TMovingSprite.DoJob : boolean;
begin
  inherited DoJob;
  if ( StayOnParentArea ) then begin
    if ( y + dy < 0 ) then dy := abs(dy);
    if ( y + dy >= Parent.ClientHeight-h ) then dy := -abs(dy);
    if ( x + dx < 0 ) then dx := abs(dx);
    if ( x + dx >= Parent.ClientWidth-w ) then dx := -abs(dx);
    CountDirection;
  end;
  y := y + dy*Speed;
  x := x + dx*Speed;
  if Assigned(FOnMove) then FOnMove(self);
  Result := true;
end;

function TMovingSprite.PaintTo(canvas: TCanvas): boolean;
var index : integer;
begin
  index := 0; if ( dx < 0 ) then index := 1;  // Kumpiko kuva pitää piirtää
  FImageList.Draw(Canvas,ix,iy,index);
  inherited PaintTo(canvas);
  Result := true;
end;

function TMovingSprite.MovingReact(sprite: TSprite): boolean;
begin
  Result := true;
end;

function TMovingSprite.React(sprite: TSprite): boolean;
begin
  Result := inherited React(sprite);
  if ( Result ) then exit;
  Result := MovingReact(sprite);
end;



end.
```

```
unit Auto;
(* Esimerkki
   1) TImageListin käytöstä
   2) TPaintBoxin perimisestä
   3) TBitMapin käytöstä (mm. peilaaminen y-akselin suhteen)
   4) Viestin lähettämisestä (Event) omistajalle
       - auto lähettää aina liikahtaessaan viestin
       - auto lähettään viestin kun se kääntyy

   Vesa Lappalainen 23.10.1997

   Changed  : 5.11.1999/vl
    + peritty TSpritestä
   Changed  : 14.11.1999/vl
    + korjattu React-metodia
    + korjattu PaintTo-metodia siten, etta suunta päätetään suunnasta, ei dx:stä
    + lisätty AutoReact perittäväksi paremmin

   Changed  : 08.09.2001/vl
    + VCL/CLX-käännös
    + entinen TAuto muutettu TLiikkuva ja TAuto peritään TLiikkuvasta *)
interface

uses Classes,sprite,moving;

type

  TAuto = class(TMovingSprite)
  private
    FRemoveOnHit : boolean;
  public
    constructor Create(AOwner:TComponent); override;
    function MovingReact(sprite: TSprite): boolean; override;
    function IsRemovable(sprite:TSprite):boolean; virtual;
    property RemoveOnHit : boolean read FRemoveOnHit write FRemoveOnHit default true;
  end;

implementation

constructor TAuto.Create(AOwner: TComponent);
begin
  FRemoveOnHit := true;
  inherited;
end;

function TAuto.IsRemovable(sprite: TSprite): boolean;
begin
  Result := false;
  if not ( sprite is TAuto ) then exit;
  if not RemoveOnHit then exit;
  if not TAuto(sprite).RemoveOnHit then exit;
  Result := true;
end;

function TAuto.MovingReact(sprite: TSprite): boolean;
begin
  Result := true;
  if not IsRemovable(sprite) then exit;
  parent.UnJoin(self,true);
  parent.UnJoin(sprite,true);
end;

end.
```



```
{-------------------------------------------------------------------------------}
{
   Unit Name: driversbrain
   Purpose  : Implement drivers brains
   Author   : Vesa Lappalainen
   Date     : 14.11.1999
   Changed  :

   ToDo     : More sence to the driver so that he does not stop AFTER the
              ligths
}
{-------------------------------------------------------------------------------}


unit driversbrain;

interface
uses sprite,auto,traffic;

type
  TDriversBrain = class(TBrain)
  private
    FOldSpeed : double;
  public
    function React(sprite:TSprite):boolean; override;
  end;


implementation

{ TDriversBrain }

function TDriversBrain.React(sprite: TSprite): boolean;
var tl : TTrafficLights; a:TAuto;
begin
  Result := false;
  if not ( Owner is TAuto ) then exit;

  if ( sprite is TTrafficLights ) then begin
    Result := true;
    a := TAuto(Owner);
    tl := TTrafficLights(sprite);
    if ( tl.State = ls_Green ) then begin
      if ( FOldSpeed = 0 ) then exit;
      a.Speed := FOldSpeed;
      FOldSpeed := 0;
      exit;
    end;
    if ( FOldSpeed <> 0 ) then exit;
    FOldSpeed := a.Speed;
    a.Speed := 0;
  end;
end;

end.
```

```
{-------------------------------------------------------------------------------}
{
   Unit Name: rusinaauto
   Purpose  : Rusinaksi törmäyksessä muuttuva auto
   Author   : Vesa Lappalainen
   Date     : 14.11.1999
   Changed  :

   ToDo     :
}
{-------------------------------------------------------------------------------}

unit rusinaauto;

interface
uses Classes,
{$ifdef CLX}
   QGraphics,
{$else}
   Graphics,
{$endif}
   sprite,auto;

type
  TRusinaState = ( rs_OK, rs_Palaa, rs_Rusina );

  TRusinaksiMuuttuvaAuto = class(TAuto)
  private
    FCounter : integer;
    FPaloAika: integer;
    FRusinaState: TRusinaState;
  public
    constructor Create(AOwner:TComponent); override;
    function PaintTo(canvas:TCanvas):boolean; override;
    function MovingReact(sprite:TSprite):boolean; override;
    function DoJob:boolean; override;
    procedure Syty;
  published
    property PaloAika : integer read FPaloAika write FPaloAika;
    property FState : TRusinaState read FRusinaState;
  end;

implementation

{ TRusinaksiMuuttuvaAuto }

function TRusinaksiMuuttuvaAuto.DoJob: boolean;
begin
  if FRusinaState = rs_OK then begin
    result := inherited DoJob;
    exit;
  end;
  Result := false;
  inc(FCounter);
  if (FCounter < Paloaika) then exit;
  Result := true;
  if ( FRusinaState = rs_Palaa ) then begin
    FCounter := 0;
    FRusinaState := rs_Rusina;
    exit;
```



```
  end;
  Parent.UnJoin(self,true);
end;

function TRusinaksiMuuttuvaAuto.PaintTo(canvas: TCanvas): boolean;
var rx,ry : integer;
begin
  if FRusinaState < rs_Rusina then inherited PaintTo(canvas);
  if FRusinaState = rs_Palaa then begin
    rx := iw * FCounter div paloaika div 2;
    ry := ih * FCounter div paloaika div 2;
    canvas.Brush.Color := clYellow;
    canvas.Ellipse(ix+iw div 2 - rx,iy,ix + iw div 2 + rx, iy + 2*ry);
  end;
  if FRusinaState = rs_Rusina then begin
    rx := iw div 2 - iw * FCounter div paloaika div 2;
    ry := ih div 2 - ih * FCounter div paloaika div 2;
    canvas.Brush.Color := clMaroon;
    canvas.Ellipse(ix+iw div 2 - rx,iy+ih-2*ry,ix + iw div 2 + rx, iy + ih);
  end;
  Result := true;
end;

function TRusinaksiMuuttuvaAuto.MovingReact(sprite: TSprite): boolean;
begin
  Result := true;
  if not IsRemovable(sprite) then exit;

  Syty;
  if ( sprite is TRusinaksiMuuttuvaAuto ) then
    TRusinaksiMuuttuvaAuto(sprite).Syty
  else
    parent.UnJoin(sprite,true);
end;

procedure TRusinaksiMuuttuvaAuto.Syty;
begin
  if  not ( FRusinaState = rs_OK ) then exit;  // jos palaa jo, ei syty enää
  FRusinaState := rs_Palaa;
end;

constructor TRusinaksiMuuttuvaAuto.Create(AOwner: TComponent);
begin
  inherited;
  PaloAika := 20;
end;

end.
```

```
{------------------------------------------------------------------------------}
{
   Unit Name: traffic
   Purpose  : Implent traffic ligths
   Author   : Vesa Lappalainen
   Date     : 14.11.1999
   Changed  :

   ToDo     :
}
{------------------------------------------------------------------------------}

unit traffic;


interface
uses Classes,
{$ifdef CLX}
  QGraphics,
{$else}
  Graphics,
{$endif}
  sprite;

type
  TLightState = (ls_Red,ls_Yellow,ls_Green);

  TTrafficLights = class(TSprite)
  private
    FState: TLightState;
    FCounter : integer;
    FInterval: integer;
    FRedProcent: integer;
    FGreenProcent: integer;
    procedure SetInterval(const Value: integer);
    procedure SetRedProcent(const Value: integer);
    procedure SetGreenProcent(const Value: integer);
  public
    constructor Create(AOwner:TComponent); override;
    function DoJob:boolean; override;
    function PaintTo(canvas:TCanvas):boolean; override;
  published
    property State : TLightState read FState;
    property Interval : integer read FInterval write SetInterval;
    property RedProcent : integer read FRedProcent write SetRedProcent;
    property GreenProcent : integer read FGreenProcent write SetGreenProcent;
  end;


implementation

{ TTrafficLights }

const lights : array[false..true,ls_Red..ls_Green] of TColor = (
  (clBlack,clBlack,clBlack),
  (clRed,clYellow,clLime)
  );

constructor TTrafficLights.Create(AOwner: TComponent);
begin
  inherited;
  FInterval := 10;
  FRedProcent := 40;
  FGreenProcent := 40;
```



```
  FCounter := 0;
  w := 30;
  h := 40;
end;

function TTrafficLights.DoJob: boolean;
var old_state : TLightState;
begin
  old_state := State;
  inc(FCounter);
  if ( FCounter >= Interval * RedProcent div 100) then FState := ls_Yellow;
  if ( FCounter >= Interval - (Interval * GreenProcent div 100) ) then FState := ls_Green;
  if ( FCounter >= Interval ) then begin FState := ls_Red; FCounter := 0; end;
  Result := old_state <> State;
end;

function TTrafficLights.PaintTo(canvas: TCanvas): boolean;
var lx,ly,r : integer; s : TLightState;
begin
  canvas.Brush.Color := clBlack;
  canvas.Rectangle(ix+iw div 2-2,iy,ix+iw div 2+2,iy+ih);
  r := ih div 10;
  lx := ix + iw div 2 - r;
  ly := iy;
  for s := ls_Red to ls_Green do begin
    canvas.Brush.Color := lights[State=s,s];
    canvas.Ellipse(lx,ly,lx+2*r,ly+2*r);
    ly := ly + 2*r;
  end;
  Result := true;
end;

procedure TTrafficLights.SetGreenProcent(const Value: integer);
begin
  FGreenProcent := Value;
end;

procedure TTrafficLights.SetInterval(const Value: integer);
begin
  FInterval := Value;
end;

procedure TTrafficLights.SetRedProcent(const Value: integer);
begin
  FRedProcent := Value;
end;

end.
```