Terrorist's Revenge

Entwicklung, Programmierung und Design eines Ego-Shooters mit OpenGL

- Anhang -

Maturaarbeit im Fach Informatik
von
David Halter (4mb)

betreuende Lehrkraft
Lukas Lippert

18. Oktober 2007

Anhang zur Maturaarbeit von David Halter

Der Anhang umfasst den Quellcode von Terrorist's Revenge. Allerdings umfasst der Anhang keine Bibliotheken. Die meisten Units (ca. 90%) wurden selbst verfasst, beim Rest habe ich zumindest grosse und wichtige Teile hinzugefügt. Ich habe versucht den Quellcode möglichst kurz zu halten, da es sehr viel ist. Dennoch kamen einige Seiten zusammen. Es sollte sich aber niemand dazu genötigt fühlen, diesen Code zu lesen.

Inhaltsverzeichnis

1. Shader

Shader, die mit "f" anfangen, sind Fragmentshader, diejenigen mit "v" sind Vertexshader.

1.1. f_blur_row

```
//this samples just the rows.

uniform sampler2D Texture0;

uniform float width;
uniform float height;
uniform float facwidth;
uniform float facheight;

const int samples = 8;
const float sum   = 3.0359;

void main(void)
{
  //Gausssche Normalverteilung:
  float sample[9];
  sample[0] = 1.0000;
  sample[1] = 0.9394;
  sample[2] = 0.7788;
  sample[3] = 0.5697;
  sample[4] = 0.3678;
  sample[6] = 0.2096;
  sample[5] = 0.1053;
  sample[7] = 0.0467;
  sample[8] = 0.0183;

 vec2 tc = vec2(gl_TexCoord[0]);
 float xp = facwidth/width;

 vec4 col = texture2D(Texture0, tc);
 vec2 tco = tc;

 col = texture2D(Texture0, vec2(tc.s-(1.0*xp), tc.t)) * sample[1];
 col = texture2D(Texture0, vec2(tc.s+(1.0*xp), tc.t)) * sample[1];

 /*col = texture2D(Texture0, vec2(tc.s-(2.0*xp), tc.t)) * sample[2];
 col += texture2D(Texture0, vec2(tc.s+(2.0*xp), tc.t)) * sample[2];

 col += texture2D(Texture0, vec2(tc.s-(3.0*xp), tc.t)) * sample[3];
 col += texture2D(Texture0, vec2(tc.s+(3.0*xp), tc.t)) * sample[3];

 col += texture2D(Texture0, vec2(tc.s-(4.0*xp), tc.t)) * sample[4];
 col += texture2D(Texture0, vec2(tc.s+(4.0*xp), tc.t)) * sample[4];

 col += texture2D(Texture0, vec2(tc.s-(5.0*xp), tc.t)) * sample[5];
```

```
 col += texture2D(Texture0, vec2(tc.s+(5.0*xp), tc.t)) * sample[5];

 col += texture2D(Texture0, vec2(tc.s-(6.0*xp), tc.t)) * sample[6];
 col += texture2D(Texture0, vec2(tc.s+(6.0*xp), tc.t)) * sample[6];

 col += texture2D(Texture0, vec2(tc.s-(7.0*xp), tc.t)) * sample[7];
 col += texture2D(Texture0, vec2(tc.s+(7.0*xp), tc.t)) * sample[7];

 col += texture2D(Texture0, vec2(tc.s-(8.0*xp), tc.t)) * sample[8];
 col += texture2D(Texture0, vec2(tc.s+(8.0*xp), tc.t)) * sample[8]; */

 //for (int i = 1; i<=samples; i++)
 //{
  //col += texture2D(Texture0, vec2(tc.s+(float(i)*xp), tc.t)) * sample[i];
  //col += texture2D(Texture0, vec2(tc.s+(float(i)*xp), tc.t));//*sample[i];
 //}
 //gl_FragColor.r = col.r/(1.0+2.0*sum);
 //gl_FragColor.g = col.g/(1.0+2.0*sum);
 //gl_FragColor.b = col.b/(1.0+2.0*sum);
 //gl_FragColor.a = 1.0;
 gl_FragColor = col/(1.0+2.0*sum);
 //gl_FragColor = texture2D(Texture0, tc.s-3.0*xp, tc.t)*0.000388 + texture2D(Texture0, tc.s-
2.0*xp, tc.t)*0.013303 + texture2D(Texture0, tc.s-xp, tc.t)*0.110979 + texture2D(Texture0,
tc.s, tc.t)*0.225079 + texture2D(Texture0, tc.s+xp, tc.t)*0.110979 + texture2D(Texture0,
tc.s+2.0*xp, tc.t)*0.013303 + texture2D(Texture0, tc.s+3.0*xp, tc.t)*0.000388;
 //gl_FragColor = gl_FragColor*3.0;
}
```

1.2. v_blur_row

```
uniform sampler2D Texture0;

void main(void)
{
 gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
 gl_TexCoord[0]= gl_MultiTexCoord0;

/*
  float sample[9];
  sample[0] = 1.0000;
  sample[1] = 0.9394;
  sample[2] = 0.7788;
  sample[3] = 0.5697;
  sample[4] = 0.3678;
  sample[6] = 0.2096;
  sample[5] = 0.1053;
  sample[7] = 0.0467;
  sample[8] = 0.0183;

 vec2 tc = vec2(gl_TexCoord[0]);
 float xp = 1.0;//facwidth/width;

 vec4 col = texture2D(Texture0, tc);
```

```
 vec2 tco = tc;

 col = texture2D(Texture0, vec2(tc.s-(1.0*xp), tc.t)) * sample[1];
 col = texture2D(Texture0, vec2(tc.s+(1.0*xp), tc.t)) * sample[1];

*/
}
```

1.3. f_blur_column

```
uniform sampler2D Texture0;

void main(void)
{
 gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
 gl_TexCoord[0]= gl_MultiTexCoord0;

/*
  float sample[9];
  sample[0] = 1.0000;
  sample[1] = 0.9394;
  sample[2] = 0.7788;
  sample[3] = 0.5697;
  sample[4] = 0.3678;
  sample[6] = 0.2096;
  sample[5] = 0.1053;
  sample[7] = 0.0467;
  sample[8] = 0.0183;

 vec2 tc = vec2(gl_TexCoord[0]);
 float xp = 1.0;//facwidth/width;

 vec4 col = texture2D(Texture0, tc);
 vec2 tco = tc;

 col = texture2D(Texture0, vec2(tc.s-(1.0*xp), tc.t)) * sample[1];
 col = texture2D(Texture0, vec2(tc.s+(1.0*xp), tc.t)) * sample[1];

*/
}
```

1.4. v_blur_column

```
uniform sampler2D Texture0;

void main(void)
{
 gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
 gl_TexCoord[0]= gl_MultiTexCoord0;

/*
  float sample[9];
  sample[0] = 1.0000;
```

```
  sample[1] = 0.9394;
  sample[2] = 0.7788;
  sample[3] = 0.5697;
  sample[4] = 0.3678;
  sample[6] = 0.2096;
  sample[5] = 0.1053;
  sample[7] = 0.0467;
  sample[8] = 0.0183;

 vec2 tc = vec2(gl_TexCoord[0]);
 float xp = 1.0;//facwidth/width;

 vec4 col = texture2D(Texture0, tc);
 vec2 tco = tc;

 col = texture2D(Texture0, vec2(tc.s-(1.0*xp), tc.t)) * sample[1];
 col = texture2D(Texture0, vec2(tc.s+(1.0*xp), tc.t)) * sample[1];

*/
}
```

1.5. f_bloom

```
//this samples just the rows.

uniform sampler2D blurredimg;
uniform sampler2D normal;

uniform float width;
uniform float height;
uniform float facwidth;
uniform float facheight;

vec4 minus = vec4(0.7, 0.7, 0.7, 0.0);

void main(void)
{
 vec2 tc = vec2(gl_TexCoord[0]);
 vec4 col = max(texture2D(blurredimg, tc) - minus, 0.0)*2.0;
 gl_FragColor = min(texture2D(normal, tc)*0.9+col, 1.0);
 //gl_FragColor = col;
 //gl_FragColor = texture2D(normal, tc);
}
```

1.6. v_bloom

```
uniform sampler2D Texture0;

void main(void)
{
 gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
 gl_TexCoord[0]= gl_MultiTexCoord0;
}
```

2. GUI-Editor

```
unit OpenGL15_MainForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,

  dglOpenGL,
  GUI,

  Menus, ExtCtrls, StdCtrls, ComCtrls, Grids, ValEdit, Buttons;

type
  TGLForm = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Edit1: TMenuItem;
    Help1: TMenuItem;
    Save1: TMenuItem;
    Load1: TMenuItem;
    Quit1: TMenuItem;
    About1: TMenuItem;
    New1: TMenuItem;
    Panel2: TPanel;
    ComboBox1: TComboBox;
    Button1: TButton;
    ValueListEditor1: TValueListEditor;
    CheckBox1: TCheckBox;
    ColorDialog1: TColorDialog;
    Button2: TButton;
    Button5: TButton;
    Label2: TLabel;
    OpenDialog1: TOpenDialog;
    Label3: TLabel;
    CheckBox2: TCheckBox;
    Edit2: TEdit;
    Label4: TLabel;
    Label5: TLabel;
    Edit3: TEdit;
    CheckBox3: TCheckBox;
    CheckBox4: TCheckBox;
    Edit4: TEdit;
    Label6: TLabel;
    CheckBox5: TCheckBox;
    CheckBox6: TCheckBox;
    CheckBox7: TCheckBox;
    Button6: TButton;
```

```
    Button7: TButton;
    Label7: TLabel;
    Edit5: TEdit;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Edit8: TEdit;
    Label11: TLabel;
    Label12: TLabel;
    Edit9: TEdit;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    Edit6: TEdit;
    Label16: TLabel;
    Edit7: TEdit;
    Label17: TLabel;
    Label18: TLabel;
    Edit11: TEdit;
    Panel3: TPanel;
    Panel4: TPanel;
    Label1: TLabel;
    Label19: TLabel;
    Edit10: TEdit;
    Label20: TLabel;
    Edit12: TEdit;
    SaveDialog1: TSaveDialog;
    Edit13: TEdit;
    Label21: TLabel;
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Panel4Click(Sender: TObject);
    procedure Panel3Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
    procedure Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure Panel1MouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Panel1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure ApplicationEventsIdle(Sender: TObject; var Done: Boolean);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private-Deklarationen }
    procedure setValues;
    procedure changeValues;
```

```
  public
    RC        : HGLRC;
    DC        : HDC;
    ShowFPS   : Boolean;
    FontBase  : GLUInt;
    StartTick : Cardinal;
    Frames    : Integer;
    FPS       : Single;
    procedure BuildFont(pFontName : String);
    procedure PrintText(pX,pY : Integer; const pText : String);
    procedure ShowText;
  end;

function CardinaltoColor(Color: Cardinal): TColor;
function ColorToCardinal(Color: TColor): Cardinal;
var
  GLForm: TGLForm;

implementation

{$R *.dfm}


function CardinaltoColor(Color: Cardinal): TColor;
begin
  result.a := 1;
  result.r := GetRValue(Color)/255;
  result.g := GetGValue(Color)/255;
  result.b := GetBValue(Color)/255;
end;

function ColorToCardinal(Color: TColor): Cardinal;
begin
  result := rgb(round(Color.r*255),round(Color.g*255),round(Color.b*255));
end;

//
============================================================
===============
//  TForm1.BuildFont
//
============================================================
===============
//  Displaylisten für Bitmapfont erstellen
//
============================================================
===============
procedure TGLForm.BuildFont(pFontName : String);
var
 Font : HFONT;
begin
  // Displaylisten für 256 Zeichen erstellen
```

```
  FontBase := glGenLists(96);
  // Fontobjekt erstellen
  Font     := CreateFont(16, 0, 0, 0, FW_MEDIUM, 0, 0, 0, ANSI_CHARSET, OUT_TT_PRECIS,
CLIP_DEFAULT_PRECIS,
                         ANTIALIASED_QUALITY, FF_DONTCARE or DEFAULT_PITCH,
PChar(pFontName));
  // Fontobjekt als aktuell setzen
  SelectObject(DC, Font);
  // Displaylisten erstellen
  wglUseFontBitmaps(DC, 0, 256, FontBase);
  // Fontobjekt wieder freigeben
  DeleteObject(Font)
end;

//
============================================================================
================
//  TForm1.PrintText
//
============================================================================
================
//  Gibt einen Text an Position x/y aus
//
============================================================================
================
procedure TGLForm.PrintText(pX,pY : Integer; const pText : String);
begin
  if (pText = '') then
    exit;
  glPushAttrib(GL_LIST_BIT);
    glRasterPos2i(pX, pY);
    glListBase(FontBase);
    glCallLists(Length(pText), GL_UNSIGNED_BYTE, PChar(pText));
  glPopAttrib;
end;

//
============================================================================
================
//  TForm1.ShowText
//
============================================================================
================
//  FPS, Hilfstext usw. ausgeben
//
============================================================================
================
procedure TGLForm.ShowText;
begin
  // Tiefentest und Texturierung für Textanzeige deaktivieren
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_TEXTURE_2D);
```

```
  // In orthagonale (2D) Ansicht wechseln
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity;
  glOrtho(0,Panel1.Width,Panel1.Height,0, -10,10);
  GUIclass.Render;
  PrintText(5,15, FloatToStr(FPS)+' fps');
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_TEXTURE_2D);
end;

//
================================================================
================
//  TForm1.FormCreate
//
================================================================
================
//  OpenGL-Initialisierungen kommen hier rein
//
================================================================
================
procedure TGLForm.FormCreate(Sender: TObject);
var
  i: integer;
begin
  // Wenn gewollt, dann hier in den Vollbildmodus wechseln
  // Muss vorm Erstellen des Kontextes geschehen, da durch den Wechsel der
  // Gerätekontext ungültig wird!
  // GoToFullscreen(1600, 1200, 32, 75);

  // OpenGL-Funtionen initialisieren
  InitOpenGL;
  // Gerätekontext holen
  DC := GetDC(Panel1.Handle);
  // Renderkontext erstellen (32 Bit Farbtiefe, 24 Bit Tiefenpuffer, Doublebuffering)
  RC := CreateRenderingContext(DC, [opDoubleBuffered], 32, 24, 0, 0, 0, 0);
  // Erstellten Renderkontext aktivieren
  ActivateRenderingContext(DC, RC);
  // Tiefenpuffer aktivieren
  glEnable(GL_DEPTH_TEST);
  // Nur Fragmente mit niedrigerem Z-Wert (näher an Betrachter) "durchlassen"
  glDepthFunc(GL_LESS);
  // Löschfarbe für Farbpuffer setzen
  glClearColor(0,0,0,0);
  // Displayfont erstellen
  BuildFont('MS Sans Serif');
  // Idleevent für Rendervorgang zuweisen
  Application.OnIdle := ApplicationEventsIdle;
  // Zeitpunkt des Programmstarts für FPS-Messung speichern
  StartTick := GetTickCount;
```

```
  // Viewport an Clientareal des Fensters anpassen
 glViewPort(0, 0, Panel1.ClientWidth, Panel1.ClientHeight);

 glShadeModel(GL_FLAT);
 glEnable(GL_TEXTURE_2D);
 glEnable(GL_BLEND);
 glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

 glEnable(GL_CULL_FACE);

 GUIclass.AddFont('data\fonts\arial10.fnt');
 GUIclass.AddFont('data\fonts\bodini10.fnt');
 GUIclass.AddFont('data\fonts\verdana12.fci', 0.6);
 GUIclass.AddFont('data\fonts\couriernew48.fci', 0.5);
 GUIclass.AddFont('data\fonts\couriernew48.fci', 0.35);
 GUiclass.AddFont('data\fonts\couriernew48.fci', 1);

 GUIclass.AddSkin('data\skins\windows\');
 GUIclass.AddSkin('data\skins\black blood\');

 GUIclass.AddText(30,30,0,0, 'Hello World', cWhite);      //schriften sind verkehrt und nicht
sichtbar wegen cullface..
 GUIclass.AddText(30,50,0,1, 'Hello World', cWhite);

 GUIclass.AddWindow(20,100,600,400,1,'Test', 'data\skins\black blood\default.tga');
 GUIclass.Windows[0].font := 3;
 GUIclass.Windows[0].Visible := true;
 GUIclass.Windows[0].Captionbar := true;
 GUIclass.Windows[0].dragevent := true;
 GUIclass.Windows[0].onClick := nil;
 GUIclass.Windows[0].AddText(10,80,'Butision', cBlack);
 GUIclass.Windows[0].Color := cWhite;

 GUIclass.Windows[0].AddButton(20,40,70,30,'Exit!');
 GUIclass.Windows[0].Buttons[0].onclick := StopExe;
 //GUIclass.Windows[0].Buttons[0].dragevent := true;
 GUIclass.Windows[0].Buttons[0].onkeydown := stopexe;

 GUIclass.Windows[0].AddButton(220,30,165,30,'Increase Alpha!');
 GUIclass.Windows[0].Buttons[1].onclick := IncreaseAlpha;

 GUIclass.Windows[0].AddButton(220,60,165,30,'Decrease Alpha!');
 GUIclass.Windows[0].Buttons[2].onclick := DecreaseAlpha;

 GUIclass.Windows[0].AddPanel(5,31,100,100);
 GUIclass.Windows[0].AddProgressBar(250,100,140,30);
 GUiclass.Windows[0].ProgressBars[0].onMouseDown := MousePostoProgressBar;
 GUIclass.Windows[0].ProgressBars[0].progress := 0.8;

 //red green blue... of the window...
 GUIclass.Windows[0].AddText(10,180,'Red:',cWhite);
 GUIclass.Windows[0].AddProgressBar(10,200,100,20);
```

```
  GUiclass.Windows[0].ProgressBars[1].onClick := SetRedwithProgressBar;
  GUiclass.Windows[0].ProgressBars[1].progress := GUiclass.Windows[0].Color.r;
  GUIclass.Windows[0].AddText(10,230,'Green:',cWhite);
  GUIclass.Windows[0].AddProgressBar(10,250,100,20);
  GUiclass.Windows[0].ProgressBars[2].onClick := SetGreenwithProgressBar;
  GUiclass.Windows[0].ProgressBars[2].progress := GUiclass.Windows[0].Color.g;
  GUIclass.Windows[0].AddText(10,280,'Blue:',cWhite);
  GUIclass.Windows[0].AddProgressBar(10,300,100,20);
  GUiclass.Windows[0].ProgressBars[3].onClick := SetBluewithProgressBar;
  GUiclass.Windows[0].ProgressBars[3].progress := GUiclass.Windows[0].Color.b;


  GUIclass.Windows[0].AddCheckbox(300,140,14,14, false);
  GUIclass.Windows[0].AddRadioButtonGroup;
  GUIclass.Windows[0].AddRadioButtonGroup;
  GUIclass.Windows[0].AddRadioButton(300,180,12,12,1,false);
  GUIclass.Windows[0].AddRadioButton(320,180,12,12,1,false);
  GUIclass.Windows[0].AddRadioButton(340,180,12,12,1,false);
  GUIclass.Windows[0].AddRadioButton(360,180,12,12,1,false);

  GUIclass.Windows[0].AddRadioButton(300,160,12,12,2,false);
  GUIclass.Windows[0].AddRadioButton(320,160,12,12,2,false);
  GUIclass.Windows[0].AddRadioButton(340,160,12,12,2,false);
  GUIclass.Windows[0].AddRadioButton(360,160,12,12,2,false);
  GUIclass.Windows[0].AddEdit(100,150,100,30,'Edit');
  GUIclass.Windows[0].CloseButton.Visible := true;
  GUIclass.Windows[0].MinimizeButton.Visible := true;
  GUIclass.Windows[0].MaximizeButton.Visible := true;

  GUIclass.Windows[0].AddImage(110,35,100,100,true,GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALP
HA,'data\fonts\verdana12.tga');
  GUIclass.Windows[0].Images[0].Color := cYellow;
  GUIclass.Windows[0].Images[0].dragevent := true;

  GUIclass.Windows[0].AddPopupMenu(150,20,'Minimize');
  GUIclass.Windows[0].PopupMenus[0].PopupMenuItems[0].onMouseUp := Minimize;
  GUIclass.Windows[0].PopUpMenus[0].TextColor := cRed;
  GUIclass.Windows[0].PopupMenus[0].font := 2;
  GUIclass.Windows[0].PopupMenus[0].AddPopUpItem('Fenster schliessen');
  GUIclass.Windows[0].PopupMenus[0].PopupMenuItems[1].onClick := CloseWindow;
  GUIclass.Windows[0].PopupMenus[0].AddPopUpItem('Stoppe das Programm');
  GUIclass.Windows[0].PopupMenus[0].PopupMenuItems[2].onMousedown := StopExe;
  GUIclass.Windows[0].PopupMenus[0].AddPopUpItem('Fenster zerstören');
  GUIclass.Windows[0].PopupMenus[0].PopupMenuItems[3].onMousedown := DestroyWindow;
  GUIclass.Windows[0].PopupMenus[0].AddPopUpItem('Butision4');
  GUIclass.Windows[0].PopUpMenu := GUIclass.Windows[0].PopUpMenus[0];

  GUIclass.Windows[0].AddCombobox(200,315,180,25,'TestCombo0');
  GUIclass.Windows[0].Comboboxes[0].AddComboboxItem('test1');
  GUIclass.Windows[0].Comboboxes[0].AddComboboxItem('test2');
  GUIclass.Windows[0].Comboboxes[0].AddComboboxItem('test3');
```

```
  GUIclass.Windows[0].AddTextField(120,200,200,100, 'Butision0' + chr(13) + 'Butision1' +
chr(13) + 'Butision2' + chr(13) + 'Diese Gui wurde gemacht um ein paar der Komponenten für
Terrorist''s Revenche bereitzustellen...',2);
  GUIclass.Windows[0].AddEditField(320,200,250,100,'Butision0' + chr(13) + 'Butision1' +
chr(13) + 'Butision2' + chr(13) + 'Diese Gui wurde gemacht um ein paar der Komponenten für
Terrorist''s Revenche bereitzustellen...',4);
  //Guiclass.Windows[0].EditFields[0].MakeNewLine(4);
  GUIclass.Windows[0].EditFields[0].MakeNewLine(0);
  GUIclass.Windows[0].EditFields[0].Deleteline(0);

  GUIclass.AddWindow(700,200,120,120,0,'Frame0');
  GUIclass.Windows[1].font := 2;
  GUIclass.Windows[1].FontColor := cYellow;
  GUIclass.Windows[1].Color := cCyan;
  GUIclass.Windows[1].visible := true;
  GUIclass.Windows[1].CaptionBar := true;
  GUIclass.Windows[1].AddButton(5,30,110,20,'Increase Alpha!');
  GUIclass.Windows[1].Buttons[0].onclick := IncreaseAlpha;
  GUIclass.Windows[1].AddButton(5,60,110,20,'Decrease Alpha!');
  GUIclass.Windows[1].Buttons[1].onclick := DecreaseAlpha;

  GUIclass.AddWindow(700,200,120,120,0,'Frame1');
  GUIclass.Windows[2].font := 2;
  GUIclass.Windows[2].FontColor := cYellow;
  GUIclass.Windows[2].Color := cWhite;
  GUIclass.Windows[2].visible := true;
  GUIclass.Windows[2].CaptionBar := true;
  GUIclass.Windows[2].AddButton(5,30,110,20,'Increase Alpha!');
  GUIclass.Windows[2].Buttons[0].onclick := IncreaseAlpha;
  GUIclass.Windows[2].AddButton(5,60,110,20,'Decrease Alpha!');
  GUIclass.Windows[2].Buttons[1].onclick := DecreaseAlpha;

  GUIclass.AddWindow(700,200,120,120,0,'Frame2');
  GUIclass.Windows[3].font := 2;
  GUIclass.Windows[3].FontColor := cGreen;
  GUIclass.Windows[3].Color := cYellow;
  GUIclass.Windows[3].visible := true;
  GUIclass.Windows[3].CaptionBar := true;
  GUIclass.Windows[3].AddButton(5,30,110,20,'Increase Alpha!');
  GUIclass.Windows[3].Buttons[0].onclick := IncreaseAlpha;
  GUIclass.Windows[3].AddButton(5,60,110,20,'Decrease Alpha!');
  GUIclass.Windows[3].Buttons[1].onclick := DecreaseAlpha;

  GUIclass.AddWindow(700,200,120,120,0,'Fensterli2 ;)');
  GUIclass.Windows[4].visible := true;
  GUIclass.Windows[4].CaptionBar := true;
  GUIclass.Windows[4].dragevent := true;
  GUIclass.Windows[4].AddButton(5,30,110,20,'Increase Alpha!');
  GUIclass.Windows[4].Buttons[0].onclick := IncreaseAlpha;
  GUIclass.Windows[4].AddButton(5,60,110,20,'Decrease Alpha!');
  GUIclass.Windows[4].Buttons[1].onclick := DecreaseAlpha;
```

```
  Guiclass.Windows[0].AddFrames(400,30,190,150,GUiclass.Windows[1]);
  Guiclass.Windows[0].Frames[0].AddFrame(Guiclass.Windows[2]);
  Guiclass.Windows[0].Frames[0].AddFrame(Guiclass.Windows[3]);

  GUIclass.AddText(300,20,1,3,'FUNKTIONIER ENDLICH!', cWhite);

  GUIclass.SaveWindow('test.gui', [0]);
  GUIclass.LoadWindow('test.gui', i);
end;

//
============================================================
===============
//  TForm1.FormDestroy
//
============================================================
===============
//  Hier sollte man wieder alles freigeben was man so im Speicher belegt hat
//
============================================================
===============
procedure TGLForm.FormDestroy(Sender: TObject);
begin
  // Renderkontext deaktiveren
  DeactivateRenderingContext;
  // Renderkontext "befreien"
  wglDeleteContext(RC);
  // Erhaltenen Gerätekontext auch wieder freigeben
  ReleaseDC(Handle, DC);
  // Falls wir im Vollbild sind, Bildschirmmodus wieder zurücksetzen
end;

//
============================================================
===============
//  TForm1.ApplicationEventsIdle
//
============================================================
===============
//  Hier wird gerendert. Der Idle-Event wird bei Done=False permanent aufgerufen
//
============================================================
===============
procedure TGLForm.ApplicationEventsIdle(Sender: TObject; var Done: Boolean);
var
  i: integer;
begin
 { // In die Projektionsmatrix wechseln
  glMatrixMode(GL_PROJECTION);
  // Identitätsmatrix laden
  glLoadIdentity;
  // Perspective, FOV und Tiefenreichweite setzen
```

```
  gluPerspective(60, ClientWidth/ClientHeight, 1, 128);

  // In die Modelansichtsmatrix wechseln
  glMatrixMode(GL_MODELVIEW);
  // Identitätsmatrix laden
  glLoadIdentity;
  // Farb- und Tiefenpuffer löschen     }
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  //ShowText;
  GUIclass.GoOrtho(Panel1.Width,Panel1.Height, -40, 40);
  GUIclass.Render;
  GUIclass.PrintFont(5,0,0,5,Pchar(inttostr(round(FPS))+ ' FPS'));
  GUIclass.PrintFont(5,40,2,3,Pchar(inttostr(GUIclass.Windows[0].RadioButtonGroups[1].whichche
cked)));
  for i := 0 to High(GUIclass.Windows) do
    GUIclass.PrintFont(20, 100+20*i,0,3,Pchar(floattostr(GUIclass.Windows[i].Z)));
  GUIclass.ExitOrtho(60,ClientWidth/ClientHeight,1,128);

  // Hinteren Puffer nach vorne bringen
  SwapBuffers(DC);

  // Windows denken lassen, das wir noch nicht fertig wären
  Done := False;

  // Nummer des gezeichneten Frames erhöhen
  inc(Frames);
  // FPS aktualisieren
  if GetTickCount - StartTick >= 500 then
  begin
    FPS       := Frames/(GetTickCount-StartTick)*1000;
    Frames    := 0;
    StartTick := GetTickCount
  end;
  sleep(10);
end;

//
================================================================
================
//  TForm1.FormKeyPress
//
================================================================
================
procedure TGLForm.FormKeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    #27 : Close;
  end;
end;

procedure TGLForm.Panel1MouseDown(Sender: TObject; Button: TMouseButton;
```

```
  Shift: TShiftState; X, Y: Integer);
begin
  case Button of
    mbLeft:     GUIclass.MouseDown(X,Y,1);
    mbRight:    GUIclass.MouseDown(X,Y,3);
    mbMiddle:   GUIclass.MouseDown(X,Y,2);
  end;
  setvalues;
end;

procedure TGLForm.Panel1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  case Button of
    mbLeft:     GUIclass.MouseUp(X,Y,1);
    mbRight:    GUIclass.MouseUp(X,Y,3);
    mbMiddle:   GUIclass.MouseUp(X,Y,2);
  end;
  setvalues;
end;

procedure TGLForm.Panel1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  GUIclass.MouseMove(X,Y);
end;

procedure TGLForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  GUIclass.KeyDown(Key);
end;

procedure TGLForm.FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  GUIclass.KeyUp(Key);
end;

//
##################################################################
###############
//
##################################################################
###############
//
================================================================
==============
//  Events vom GUI - Editor
//
================================================================
==============
//
```

```
##########################################################################
###############
//
##########################################################################
###############

procedure TGLForm.Button1Click(Sender: TObject);  //Add Button
var
  x,y,width, height, skin: integer;
  name: string;
  z: single;
  i,j: integer;
begin
  x          := strtoint(   ValueListEditor1.Cells[1, 1]);   //x
  y          := strtoint(   ValueListEditor1.Cells[1, 2]);   //y
  width       := strtoint(   ValueListEditor1.Cells[1, 4]);   //width
  height      := strtoint(   ValueListEditor1.Cells[1, 5]);   //height
  name        :=            ValueListEditor1.Cells[1, 6];    //caption
  skin       := strtoint(   ValueListEditor1.Cells[1, 9]);   //skin
  case Combobox1.ItemIndex of
    0: begin
         OpenDialog1.Title := 'Choose an Image';
         OpenDialog1.InitialDir := 'C:\Dokumente und
Einstellungen\David\Desktop\matura\gui\GUIEditor';
         OpenDialog1.Execute;
         i := Guiclass.AddWindow(x,y,width,height,skin,name,OpenDialog1.FileName);
         GUIclass.Windows[i].Visible := true;
         GUIclass.active := GUIclass.Windows[i];
       end;
    1:  begin
         i := Guiclass.active.Parent.AddButton(x,y,width,height,name);
         GUIclass.active := GUIclass.active.Parent.Buttons[i];
       end;
    2:  begin
         i := Guiclass.active.Parent.AddText(x,y,name,cwhite);
         GUIclass.active := GUIclass.active.Parent.Text[i];
       end;
    3:  begin
         i := Guiclass.active.Parent.AddCheckbox(x,y,width,height,false);
         GUIclass.active := GUIclass.active.Parent.Checkboxes[i];
       end;
    4:  begin
         i := Guiclass.active.Parent.AddProgressBar(x,y,width,height);
         GUIclass.active := GUIclass.active.Parent.ProgressBars[i];
       end;
    5:  begin
         i := Guiclass.active.Parent.AddPanel(x,y,width,height);
         GUIclass.active := GUIclass.active.Parent.Panels[i];
       end;
    6:  begin
         i := Guiclass.active.Parent.AddEdit(x,y,width,height,name);
         GUIclass.active := GUIclass.active.Parent.Edits[i];
```

```
      end;
   7:  begin
        i := Guiclass.active.Parent.AddRadioButton(x,y,width,height,0,false);
        GUIclass.active := GUIclass.active.Parent.RadioButtons[i];
      end;
   8:  begin
        i := Guiclass.active.Parent.AddEditField(x,y,width,height,name);
        GUIclass.active := GUIclass.active.Parent.EditFields[i];
      end;
   9:  begin
        i := Guiclass.active.Parent.AddTextField(x,y,width,height,name);
        GUIclass.active := GUIclass.active.Parent.TextFields[i];
      end;
   10: begin
        i := Guiclass.active.Parent.AddPopupMenu(width,height,name);
        GUIclass.active.PopUpMenu := GUIclass.active.Parent.PopUpMenus[i];
        GUIclass.active := GUIclass.active.Parent.PopUpMenus[i];
      end;
   11: begin
        i := Guiclass.active.Parent.AddCombobox(x,y,width,height,name);
        GUIclass.active := GUIclass.active.Parent.Comboboxes[i];
      end;
   12: begin
        j := Guiclass.AddWindow(0,0,0,0,0,'');
        i := Guiclass.active.Parent.AddFrames(x,y,width,height,guiclass.Windows[j]);
        GUIclass.active := GUIclass.active.Parent.Frames[i];
      end;
   13: begin
        OpenDialog1.Title := 'Choose an Image';
        OpenDialog1.Execute;
        i := Guiclass.active.Parent.AddImage(x,y,width,height, Checkbox2.Checked,
strtoint(Edit2.Text), strtoint(Edit3.Text), OpenDialog1.FileName);
        GUIclass.active := GUIclass.active.Parent.Images[i];
      end;
   14: GUIclass.active.Parent.AddRadioButtonGroup;

   15: if GUiclass.active.objecttype = 14 then
      begin
        GUIclass.active.Parent.Comboboxes[GUIclass.active.Index].AddComboBoxItem(name);
      end;
   16: if GUiclass.active.objecttype = 13 then
      begin
        GUIclass.active.Parent.PopupMenus[GUIclass.active.Index].AddPopUpItem(name);
      end;
   17: if GUIclass.active.objecttype = 15 then
      begin
        j := Guiclass.AddWindow(0,0,0,0,0,'');
        GUIclass.active.Parent.Frames[GUIclass.active.Index].AddFrame(GUIclass.Windows[j]);
        GUIclass.active := GUIclass.Windows[j];
      end;
  end;
  //z := strtofloat(ValueListEditor1.Cells[1, 3]);
```

```
  //changeValues;
  //GUIclass.active.Z := Z;
end;

procedure TGLForm.setValues;
begin
  try
    ValueListEditor1.Cells[1, 1] := inttostr(Guiclass.active.X);   //x
  except
    showmessage('Probleme mit setValues');
    Exit;
  end;
  ValueListEditor1.Cells[1, 2] := inttostr(Guiclass.active.y);   //y
  ValueListEditor1.Cells[1, 3] := floattostr(Guiclass.active.z);   //z
  ValueListEditor1.Cells[1, 4] := inttostr(Guiclass.active.width);   //width
  ValueListEditor1.Cells[1, 5] := inttostr(Guiclass.active.height);   //height
  ValueListEditor1.Cells[1, 6] := Guiclass.active.name;    //name
  ValueListEditor1.Cells[1, 7] := inttostr(Guiclass.active.index);   //index
  ValueListEditor1.Cells[1, 8] := inttostr(Guiclass.active.objecttype);   //objecttype
  ValueListEditor1.Cells[1, 9] := inttostr(Guiclass.active.skin);         //skin
  ValueListEditor1.Cells[1, 10] := inttostr(Guiclass.active.font);     //font
  Checkbox1.Checked := Guiclass.active.dragevent;

  case GUIclass.active.objecttype of
    0: begin    //Button
         Edit5.Text := inttostr(guiclass.active.Parent.Buttons[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.Buttons[guiclass.active.Index].plusy);
         ValueListEditor1.Cells[1, 6] :=
guiclass.active.Parent.Buttons[guiclass.active.Index].Caption;    //caption
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Buttons[guiclass.active.Index].TextColor);
       end;
    1: begin    //Text
         ValueListEditor1.Cells[1, 6] := guiclass.active.Parent.Text[guiclass.active.Index].Text;
//caption
         Panel3.Color := ColortoCardinal(guiclass.active.Parent.Text[guiclass.active.Index].Color);
       end;
    3: begin    //Progressbar
         Edit7.Text :=
floattostr(guiclass.active.Parent.Progressbars[guiclass.active.Index].progress);
       end;
    4: begin    //Edit
         Edit5.Text := inttostr(guiclass.active.Parent.Edits[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.Edits[guiclass.active.Index].plusy);
         ValueListEditor1.Cells[1, 6] := guiclass.active.Parent.Edits[guiclass.active.Index].Text;
//caption
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Edits[guiclass.active.Index].TextColor);
       end;
    6: begin    //RadioButton
         Edit11.Text := inttostr(guiclass.active.Parent.RadioButtons[guiclass.active.Index].Group);
       end;
```

```
    7: begin    //Window
         Checkbox3.Checked := GUIclass.active.Parent.background;
         Checkbox4.Checked := GUIclass.active.Parent.captionbar;
         Checkbox5.Checked := GUIclass.active.Parent.MinimizeButton.Visible;
         Checkbox6.Checked := GUIclass.active.Parent.MaximizeButton.Visible;
         Checkbox7.Checked := GUIclass.active.Parent.CloseButton.Visible;
         Edit4.Text := inttostr(GUIclass.active.Parent.CaptionBarHeight);
         Edit10.Text := floattostr(GUIclass.active.Parent.Color.A);
         ValueListEditor1.Cells[1, 6] := GUIclass.active.Parent.Caption;    //caption
         Panel4.Color := ColortoCardinal(GUIclass.active.Parent.Color);
         Panel3.Color := ColortoCardinal(GUIclass.active.Parent.FontColor);
         Edit13.Text := GUIclass.active.Parent.graphicpath;
       end;
   11: begin    //Editfield
         Edit5.Text := inttostr(guiclass.active.Parent.Editfields[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.Editfields[guiclass.active.Index].plusy);
         Edit9.Text := inttostr(guiclass.active.Parent.Editfields[guiclass.active.Index].space);
         ValueListEditor1.Cells[1, 6] := guiclass.active.Parent.Editfields[guiclass.active.Index].Text;
//caption
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Editfields[guiclass.active.Index].TextColor);
       end;
   12: begin    //Textfield
         Edit9.Text := inttostr(guiclass.active.Parent.Textfields[guiclass.active.Index].space);
         ValueListEditor1.Cells[1, 6] := guiclass.active.Parent.Textfields[guiclass.active.Index].Text;
//caption
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Textfields[guiclass.active.Index].TextColor);
       end;
   13: begin    //PopUp
         Edit5.Text := inttostr(guiclass.active.Parent.PopUpMenus[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.PopUpMenus[guiclass.active.Index].plusy);
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.PopUpMenus[guiclass.active.Index].TextColor);
       end;
   14: begin    //Combobox
         Edit5.Text := inttostr(guiclass.active.Parent.Comboboxes[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.Comboboxes[guiclass.active.Index].plusy);
         Edit9.Text := inttostr(guiclass.active.Parent.Comboboxes[guiclass.active.Index].space);
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Comboboxes[guiclass.active.Index].TextColor);
       end;
   15: begin    //Frames
         Edit5.Text := inttostr(guiclass.active.Parent.Frames[guiclass.active.Index].plusx);
         Edit8.Text := inttostr(guiclass.active.Parent.Frames[guiclass.active.Index].plusy);
         Edit4.Text :=
inttostr(guiclass.active.Parent.Frames[guiclass.active.Index].CaptionBarHeight);
         Edit6.Text := inttostr(guiclass.active.Parent.Frames[guiclass.active.Index].Tabwidth);
         Panel3.Color :=
ColortoCardinal(guiclass.active.Parent.Frames[guiclass.active.Index].TextColor);
       end;
   16: begin    //Image
```

```
        Edit2.Text := inttostr(GUIclass.active.Parent.Images[GUIclass.active.index].sfactor);
        Edit3.Text := inttostr(GUIclass.active.Parent.Images[GUIclass.active.index].dfactor);
        Checkbox2.Checked := GUIclass.active.Parent.Images[GUIclass.active.index].blending;
        Panel4.Color :=
ColortoCardinal(guiclass.active.Parent.Images[guiclass.active.Index].Color);
        Edit13.Text := GUIclass.active.Parent.Images[guiclass.active.Index].graphicpath;
      end;
   30: begin
        ValueListEditor1.Cells[1, 6] :=
GUIclass.active.Parent.PopUpMenus[GUIclass.active.tag].PopUpMenuItems[GUIclass.active.index].
Caption;
      end;
   31: begin
        ValueListEditor1.Cells[1, 6] :=
GUIclass.active.Parent.Comboboxes[GUIclass.active.tag].ComboboxItems[GUIclass.active.index].
Caption;
      end;
  end;

end;

procedure TGLForm.changeValues;
var
  _x,_y,_width, _height, {_index, _objecttype,} _font, _skin: integer;
  _z: double;
  _name: string;
begin
  //index und objecttype sind readonly und werden nicht verändert.
  _x          := strtoint(   ValueListEditor1.Cells[1, 1]);   //x
  _y          := strtoint(   ValueListEditor1.Cells[1, 2]);   //y
  _z          := strtofloat( ValueListEditor1.Cells[1, 3]);   //z
  _width       := strtoint(   ValueListEditor1.Cells[1, 4]);   //width
  _height      := strtoint(   ValueListEditor1.Cells[1, 5]);   //height
  //_name        :=            ValueListEditor1.Cells[1, 6];    //caption
  //_index       := strtoint(   ValueListEditor1.Cells[1, 7]);   //index
  //_objecttype  := strtoint(   ValueListEditor1.Cells[1, 8]);   //objecttype
  _skin        := strtoint(   ValueListEditor1.Cells[1, 9]);  //skin
  _font        := strtoint(   ValueListEditor1.Cells[1, 10]);   //font

  with Guiclass.active do
  begin
    x := _x;
    y := _y;
    z := _z;
    width := _width;
    height := _height;
    //index := _index;
    //objecttype := _objecttype;
    //name := _name;
    skin := _skin;
    font := _font;
    dragevent := Checkbox1.Checked;
```

```
  end;

  case GUIclass.active.objecttype of
    0: begin    //Button
         guiclass.active.Parent.Buttons[guiclass.active.Index].plusx := strtoint(Edit5.Text);
         guiclass.active.Parent.Buttons[guiclass.active.Index].plusy := strtoint(Edit8.Text);
         guiclass.active.Parent.Buttons[guiclass.active.Index].Caption := ValueListEditor1.Cells[1,
6];    //caption
         guiclass.active.Parent.Buttons[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
       end;
    1: begin    //Text
         guiclass.active.Parent.Text[guiclass.active.Index].Text := ValueListEditor1.Cells[1, 6];
//caption
         guiclass.active.Parent.Text[guiclass.active.Index].Color := CardinalToColor(Panel3.Color);
       end;
    3: begin    //Progressbar
         guiclass.active.Parent.Progressbars[guiclass.active.Index].progress :=
strtofloat(Edit7.Text);
       end;
    4: begin    //Edit
         guiclass.active.Parent.Edits[guiclass.active.Index].plusx := strtoint(Edit5.Text);
         guiclass.active.Parent.Edits[guiclass.active.Index].plusy := strtoint(Edit8.Text);
         guiclass.active.Parent.Edits[guiclass.active.Index].Text := ValueListEditor1.Cells[1, 6];
//caption
         guiclass.active.Parent.Edits[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
       end;
    6: begin    //RadioButton
         guiclass.active.Parent.RadioButtons[guiclass.active.Index].Group := strtoint(Edit11.Text);
       end;
    7: begin    //Window
         GUIclass.active.Parent.background := Checkbox3.Checked;
         GUIclass.active.Parent.captionbar := Checkbox4.Checked;
         GUIclass.active.Parent.MinimizeButton.Visible := Checkbox5.Checked;
         GUIclass.active.Parent.MaximizeButton.Visible := Checkbox6.Checked;
         GUIclass.active.Parent.CloseButton.Visible := Checkbox7.Checked;
         GUIclass.active.Parent.CaptionBarHeight := strtoint(Edit4.Text);
         GUIclass.active.Parent.Caption := ValueListEditor1.Cells[1, 6];    //caption
         GUIclass.active.Parent.Color := CardinalToColor(Panel4.Color);
         GUIclass.active.Parent.FontColor := CardinalToColor(Panel3.Color);
         GUIclass.active.Parent.Color.A := strtofloat(Edit10.Text);
         GUIclass.active.Parent.ChangeCaptionBarButtons;
         GUIclass.active.Parent.graphicpath := Edit13.Text;
       end;
   11: begin    //Editfield
         guiclass.active.Parent.Editfields[guiclass.active.Index].plusx := strtoint(Edit5.Text);
         guiclass.active.Parent.Editfields[guiclass.active.Index].plusy := strtoint(Edit8.Text);
         guiclass.active.Parent.Editfields[guiclass.active.Index].space := strtoint(Edit9.Text);
         guiclass.active.Parent.Editfields[guiclass.active.Index].Text := ValueListEditor1.Cells[1, 6];
//caption
         guiclass.active.Parent.Editfields[guiclass.active.Index].TextColor :=
```

```
CardinalToColor(Panel3.Color);
      end;
   12: begin    //Textfield
        guiclass.active.Parent.Textfields[guiclass.active.Index].space := strtoint(Edit9.Text);
        guiclass.active.Parent.Textfields[guiclass.active.Index].Text := ValueListEditor1.Cells[1, 6];
//caption
        guiclass.active.Parent.Textfields[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
      end;
   13: begin    //PopUp
        guiclass.active.Parent.PopUpMenus[guiclass.active.Index].plusx := strtoint(Edit5.Text);
        guiclass.active.Parent.PopUpMenus[guiclass.active.Index].plusy := strtoint(Edit8.Text);
        guiclass.active.Parent.PopUpMenus[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
      end;
   14: begin    //Combobox
        guiclass.active.Parent.Comboboxes[guiclass.active.Index].plusx := strtoint(Edit5.Text);
        guiclass.active.Parent.Comboboxes[guiclass.active.Index].plusy := strtoint(Edit8.Text);
        guiclass.active.Parent.Comboboxes[guiclass.active.Index].space := strtoint(Edit9.Text);
        guiclass.active.Parent.Comboboxes[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
      end;
   15: begin    //Frames
        guiclass.active.Parent.Frames[guiclass.active.Index].plusx := strtoint(Edit5.Text);
        guiclass.active.Parent.Frames[guiclass.active.Index].plusy := strtoint(Edit8.Text);
        guiclass.active.Parent.Frames[guiclass.active.Index].CaptionBarHeight :=
strtoint(Edit4.Text);
        guiclass.active.Parent.Frames[guiclass.active.Index].Tabwidth := strtoint(Edit6.Text);
        guiclass.active.Parent.Frames[guiclass.active.Index].TextColor :=
CardinalToColor(Panel3.Color);
      end;
   16: begin                                    //Image
        GUIclass.active.Parent.Images[GUIclass.active.index].sfactor := strtoint(Edit2.Text);
        GUIclass.active.Parent.Images[GUIclass.active.index].dfactor := strtoint(Edit3.Text);
        GUIclass.active.Parent.Images[GUIclass.active.index].blending := Checkbox2.Checked;
        guiclass.active.Parent.Images[guiclass.active.Index].Color :=
CardinalToColor(Panel4.Color);
        GUIclass.active.Parent.Images[guiclass.active.Index].graphicpath := Edit13.Text;
      end;
   30: begin
        GUIclass.active.Parent.PopUpMenus[GUIclass.active.tag].PopUpMenuItems[GUIclass.active.
index].Caption := ValueListEditor1.Cells[1, 6];
      end;
   31: begin
        GUIclass.active.Parent.Comboboxes[GUIclass.active.tag].ComboboxItems[GUIclass.active.i
ndex].Caption := ValueListEditor1.Cells[1, 6];
      end;
  end;


end;
```

```
procedure TGLForm.Button2Click(Sender: TObject); //change
begin
  changeValues;
end;

procedure TGLForm.Button5Click(Sender: TObject);     //Delete
var
  i: integer;
begin
  if GUIclass.active.objecttype = 7 then
    GUIclass.DeleteWindow(GUIclass.active.index)
    else if GUIclass.active.objecttype = 15 then
    begin                          //evtl was ändern...
      GUIclass.active.Visible := false;
      GUIclass.active.Used := false;
      for i := 1 to High(GUIclass.active.Parent.Frames) do
      begin
        GUIclass.active.Parent.Frames[GUIclass.active.Index].Frames[i].Window.FreeComponents;
        GUIclass.active.Parent.Frames[GUIclass.active.Index].Frames[i].Window.Free;
      end;
      setlength(GUIclass.active.Parent.Frames[0].Frames, 1);
      GUIclass.active.Parent.Frames[GUIclass.active.Index].Frames[0].Window.used := false;
      GUIclass.active.Parent.Frames[GUIclass.active.Index].Frames[0].Window.visible := false;
    end else
      begin
        GUIclass.active.Visible := false;
        GUIclass.active.Used := false;
      end;
end;

procedure TGLForm.Panel3Click(Sender: TObject);
begin
  ColorDialog1.Color := Panel3.Color;
  ColorDialog1.Execute;
  Panel3.Color := ColorDialog1.Color;
end;

procedure TGLForm.Panel4Click(Sender: TObject);
begin
  ColorDialog1.Color := Panel4.Color;
  ColorDialog1.Execute;
  Panel4.Color := ColorDialog1.Color;
end;

procedure TGLForm.Button7Click(Sender: TObject);
var
  w: array of integer;
  i: integer;
  lastch: integer;
begin
  SaveDialog1.Execute;
  if SaveDialog1.FileName <> '' then
```

```
  begin
    setlength(w, 0);
    lastch := 0;
    for i := 0 to length(Edit12.Text) do
      if Edit12.Text[i] = ',' then
      begin
        setlength(w, length(w) + 1);
        w[High(w)] := strtoint(copy(Edit12.Text, lastch+1, i-lastch-1));
        lastch := i;
      end;
    GUIclass.SaveWindow(SaveDialog1.FileName, w)
  end;
end;

procedure TGLForm.Button6Click(Sender: TObject);      //load
var
  w: array of integer;
begin
  OpenDialog1.Execute;
  if OpenDialog1.FileName <> '' then
    GUIclass.LoadWindow(OpenDialog1.FileName, w)
end;

end.
  {
  strtoint(   ValueListEditor1.Cells[1, 1]);   //x
  strtoint(   ValueListEditor1.Cells[1, 2]);   //y
  strtofloat( ValueListEditor1.Cells[1, 3]);   //z
  strtoint(   ValueListEditor1.Cells[1, 4]);   //width
  strtoint(   ValueListEditor1.Cells[1, 5]);   //height
              ValueListEditor1.Cells[1, 6];    //caption
  strtoint(   ValueListEditor1.Cells[1, 7]);   //index
  strtoint(   ValueListEditor1.Cells[1, 8]);   //objecttype
  strtoint(   ValueListEditor1.Cells[1, 9]);   //skin
  strtoint(   ValueListEditor1.Cells[1, 10]);  //font
  }
```

3. Basics

```
unit Basics;

interface

uses
  gl3ds,
  dglOpenGL,
  SysUtils;

const
  DEGTORAD = 0.017453292519943295769236907684886;
  RADTODEG = 57.295779513082320876798154814105;

type
  PVector3i = ^TVector3i;
  TVector3i = record
    x,y,z: integer;
  end;

  PVector2i = ^TVector2i;
  TVector2i = record
    x,y: integer;
  end;

  PVector3d = ^TVector3d;
  TVector3d = record
    x,y,z: double;
  end;

  PVector2d = ^TVector2d;
  TVector2d = record
    x,y: double;
  end;

  PVector2f = ^TVector2f;
  TVector2f = record
    x,y: single;
  end;

  PVector3f = ^TVector3f;
  TVector3f = record
    x,y,z: single;
  end;

  TColor4i = record
    r,g,b,a: integer;      // [0..255]
  end;

  TColor3i = record
    r,g,b: integer;      // [0..255]
```

```
 end;

 TColor4d = record
  r,g,b,a: double;      // [0..1]
 end;

 TColor3d = record
  r,g,b: double;      // [0..1]
 end;

 TColor4f = record
  r,g,b,a: single;      // [0..1]
 end;

 TColor3f = record
  r,g,b: single;      // [0..1]
 end;

 //---------------- specially for Terrorist's Revenche! ----------------------
 TPlane = record
  a, b, c, d: Single;                    //ax + by + cz + d = 0
 end;

 TPlayerPosition = record             //for network
  x,y,z: single;
  lookanglex, lookangley: single;
 end;

 PFullVertex = ^TFullVertex;
 TFullVertex = record
  vertex:    TVector3f;
  normal:    TVector3f;
  texCoord:  TVector2f;
 end;

 {PPolygon    = record
  v        : array [0..2] of PFullVertex;
 end;   }

 PPolygon = ^TPolygon;

 TPolygon = record
  Vertexes: array [0..2] of TFullVertex;
  d:        single;  //Ebenengleichung:   alle Punkte x mit:    Skalar(normal,Vektor x):=Triangle.d
  //plane:    TPlane;
  material: word;
 end;

 TPolyList = array of PPolygon;

 PSpheref = ^TSpheref;
 TSpheref = record
```

```
    center:  TVector3f;
    radius:  single;
  end;

  TRayf = record
    Origin:     TVector3f;
    Direction:  TVector3f;       //normalized (should be)
  end;

  //Radian vs. Degree
  function DegreetoRadian(const i: integer): single;              overload;
  function DegreetoRadian(const d: double): double;                overload;
  procedure RadiantoDegree(const rad: double; var deg: integer);    overload;
  procedure DegreetoRadian(const rad: double; var deg: double);     overload;

  //vector geometry
  function skalar(const VectorA,VectorB: TVector3d): double;             overload;
  function skalar(const VectorA,VectorB: TVector3f): single;             overload;
  function minusVector(const VectorA,VectorB: TVector3d): TVector3d;      overload;
  function minusVector(const VectorA,VectorB: TVector3f): TVector3f;      overload;
  function plusVector(const VectorA,VectorB: TVector3d): TVector3d;       overload;
  function plusVector(const VectorA,VectorB: TVector3f): TVector3f;       overload;
  function malVector(const Vector: TVector3d; factor: double): TVector3d; overload;
  function malVector(const Vector: TVector3f; factor: double): TVector3f; overload;
  function Magnitude(const Vector : TVector3d) : double;                  overload;
  function Magnitude(const Vector : TVector3f) : single;                  overload;
  function Normalize(const Vector: TVector3d): TVector3d;                 overload;
  function Normalize(const Vector: TVector3f): TVector3f;                 overload;
  function CrossProduct(const VectorA,VectorB : TVector3d) : TVector3d;   overload;
  function CrossProduct(const VectorA,VectorB : TVector3f) : TVector3f;   overload;
  function GetNormal(Triangle : PPolygon) : TVector3f;
  procedure CalculateNormals(Triangle : PPolygon);
  function CalculatePlane(Triangle: PPolygon): TPlane;
  function Vec3d(_X, _Y, _Z: double): TVector3d;
  function Vec3f(_X, _Y, _Z: single): TVector3f;
  function Dist(const Vec1, Vec2: TVector3d): double; overload;
  function Dist(const Vec1, Vec2: TVector3f): single; overload;

  //colisions
  function Col_Plane_Ray(const PlaneNormal: TVector3f; planeD: single; const Ray: TRayf):
TVector3f;
  function Col_Plane_Point(const Normal, Point: TVector3f; planeD: single): boolean;
  function Col_Triangle_Sphere(const Triangle: PPolygon; const Sphere: TSpheref): boolean;
  function Col_Triangle_Point(const Triangle: PPolygon; const Point: TVector3f): boolean;
  function Col_Triangle_Ray_Return_Dist(const Triangle: PPolygon; const Ray: TRayf; var Dist:
single): boolean;
  function Col_Triangle_Ray(const Triangle: PPolygon; const Ray: TRayf; Maxdist: single):
boolean;
  function Col_Sphere_Ray(const Sphere: TSpheref; const Ray: TRayf): boolean;

  //OpenGL
  function CreateEmptyTexture(Width, Height, bpp: integer): Cardinal;
```

```
  //Misc
  function isInteger(s: String): boolean;
  function isFloat(s: String): boolean;
  function VectorToString(const Vec: TVector3f): string;
  function PolygonToString(Poly: PPolygon): string;

implementation

//----------------------------------------------------------------------------
//  Radian and Degree Calculations
//----------------------------------------------------------------------------
function DegreetoRadian(const i: integer): single; overload;
begin
  result := i*Pi/180;
end;

function DegreetoRadian(const d: double): double; overload;
begin
  result := d*Pi/180;
end;

procedure RadiantoDegree(const rad: double; var deg: integer); overload;
begin
  deg := round(rad*180/Pi);
end;

procedure DegreetoRadian(const rad: double; var deg: double); overload;
begin
  deg := rad*180/Pi
end;

//----------------------------------------------------------------------------
//  Vector Geometry
//----------------------------------------------------------------------------
function skalar(const VectorA,VectorB: TVector3d): double;
begin
  result:= VectorA.x*VectorB.x + VectorA.y*VectorB.y + VectorA.z*VectorB.z;
end;

function skalar(const VectorA,VectorB: TVector3f): single;
begin
  result:= VectorA.x*VectorB.x + VectorA.y*VectorB.y + VectorA.z*VectorB.z;
end;

function minusVector(const VectorA,VectorB: TVector3d): TVector3d;
begin
  result.x := VectorA.x - VectorB.x;
  result.y := VectorA.y - VectorB.y;
  result.z := VectorA.z - VectorB.z;
end;
```

```
function minusVector(const VectorA,VectorB: TVector3f): TVector3f;
begin
  result.x := VectorA.x - VectorB.x;
  result.y := VectorA.y - VectorB.y;
  result.z := VectorA.z - VectorB.z;
end;

function plusVector(const VectorA,VectorB: TVector3d): TVector3d;
begin
  result.x := VectorA.x + VectorB.x;
  result.y := VectorA.y + VectorB.y;
  result.z := VectorA.z + VectorB.z;
end;

function plusVector(const VectorA,VectorB: TVector3f): TVector3f;
begin
  result.x := VectorA.x + VectorB.x;
  result.y := VectorA.y + VectorB.y;
  result.z := VectorA.z + VectorB.z;
end;

function malVector(const Vector: TVector3d; factor: double): TVector3d;
begin
  result.x:=Vector.x*factor;
  result.y:=Vector.y*factor;
  result.z:=Vector.z*factor;
end;

function malVector(const Vector: TVector3f; factor: double): TVector3f;
begin
  result.x:=Vector.x*factor;
  result.y:=Vector.y*factor;
  result.z:=Vector.z*factor;
end;

function Magnitude(const Vector : TVector3d) : double;
begin
  {Returns the length of the Vector}
   result := sqrt(Vector.X * Vector.X+
                  Vector.Y * Vector.Y+
                  Vector.Z * Vector.Z);
end;

function Magnitude(const Vector : TVector3f) : Single;
begin
  {Returns the length of the Vector}
   result := sqrt(Vector.X * Vector.X+
                  Vector.Y * Vector.Y+
                  Vector.Z * Vector.Z);
end;

function Normalize(const Vector: TVector3d): TVector3d;
```

```
var
  Length: single;
begin
  Length := sqrt(sqr (Vector.x) + sqr(Vector.y) + sqr(Vector.z));
  if Length = 0 then
    Length := 1;
  result.x := Vector.x / Length;
  result.y := Vector.y / Length;
  result.z := Vector.z / Length;
end;

function Normalize(const Vector: TVector3f): TVector3f;
var
  Length: single;
begin
  Length := sqrt(sqr (Vector.x) + sqr(Vector.y) + sqr(Vector.z));
  if Length = 0 then
    Length := 1;
  result.x := Vector.x / Length;
  result.y := Vector.y / Length;
  result.z := Vector.z / Length;
end;

function CrossProduct(const VectorA,VectorB : TVector3d) : TVector3d;
begin
   Result.X := VectorA.Y * VectorB.Z - VectorA.Z * VectorB.Y;
   Result.Y := VectorA.Z * VectorB.X - VectorA.X * VectorB.Z;
   Result.Z := VectorA.X * VectorB.Y - VectorA.Y * VectorB.X;
end;

function CrossProduct(const VectorA,VectorB : TVector3f) : TVector3f;
begin
   Result.X := VectorA.Y * VectorB.Z - VectorA.Z * VectorB.Y;
   Result.Y := VectorA.Z * VectorB.X - VectorA.X * VectorB.Z;
   Result.Z := VectorA.X * VectorB.Y - VectorA.Y * VectorB.X;
end;

function GetNormal(Triangle : PPolygon) : TVector3f;
begin
  result :=    Normalize(
                 CrossProduct(
                       minusVector(Triangle^.Vertexes[1].Vertex, Triangle^.Vertexes[0].Vertex),
                       minusVector(Triangle^.Vertexes[2].Vertex, Triangle^.Vertexes[1].Vertex)
                          )
                    );
end;

function CalculatePlane(Triangle: PPolygon): TPlane;
begin
  with Triangle^ do
  begin
    result.a :=  Vertexes[0].Vertex.y * (Vertexes[1].Vertex.z - Vertexes[2].Vertex.z) +
```

```
            Vertexes[1].Vertex.y * (Vertexes[2].Vertex.z - Vertexes[0].Vertex.z) +
            Vertexes[2].Vertex.y * (Vertexes[0].Vertex.z - Vertexes[1].Vertex.z);
   result.b :=  Vertexes[0].Vertex.z * (Vertexes[1].Vertex.x - Vertexes[2].Vertex.x) +
            Vertexes[1].Vertex.z * (Vertexes[2].Vertex.x - Vertexes[0].Vertex.x) +
            Vertexes[2].Vertex.z * (Vertexes[0].Vertex.x - Vertexes[1].Vertex.x);
   result.c :=  Vertexes[0].Vertex.x * (Vertexes[1].Vertex.y - Vertexes[2].Vertex.y) +
            Vertexes[1].Vertex.x * (Vertexes[2].Vertex.y - Vertexes[0].Vertex.y) +
            Vertexes[2].Vertex.x * (Vertexes[0].Vertex.y - Vertexes[1].Vertex.y);
   result.d :=  -(  Vertexes[0].Vertex.x * (Vertexes[1].Vertex.y * Vertexes[2].Vertex.z -
Vertexes[2].Vertex.y * Vertexes[1].Vertex.z) +
                Vertexes[1].Vertex.x * (Vertexes[2].Vertex.y * Vertexes[0].Vertex.z -
Vertexes[0].Vertex.y * Vertexes[2].Vertex.z) +
                Vertexes[2].Vertex.x * (Vertexes[0].Vertex.y * Vertexes[1].Vertex.z -
Vertexes[1].Vertex.y * Vertexes[0].Vertex.z)    );
  end;
end;

procedure CalculateNormals(Triangle : PPolygon);
var
  normal: TVector3f;
  var
    i: integer;
begin
  normal := GetNormal(Triangle);
  for i := 0 to 2 do
    Triangle.Vertexes[i].normal := normal;
  // Ebenengleichung:   alle Punkte x mit:    Skalar(normal,Vektor x):=Triangle.d
  Triangle.d := skalar(normal,triangle.vertexes[0].vertex);
end;

function Vec3f(_X, _Y, _Z: single): TVector3f;
begin
  result.x := _X;
  result.y := _Y;
  result.z := _Z;
end;

function Vec3d(_X, _Y, _Z: double): TVector3d;
begin
  result.x := _X;
  result.y := _Y;
  result.z := _Z;
end;



function Dist(const Vec1, Vec2: TVector3d): double;
begin
  result := Magnitude(MinusVector(Vec1,Vec2));
end;

function Dist(const Vec1, Vec2: TVector3f): single;
```

```
begin
  result := Magnitude(MinusVector(Vec1, Vec2));
end;

//----------------------------------------------------------------------------
//  Collisions
//----------------------------------------------------------------------------
function Col_Plane_Ray(const PlaneNormal: TVector3f; planeD: single; const Ray: TRayf):
TVector3f;    //result: the Point of intersection...
var
  t: double;
  dist: single;
  P: TVector3f;
  divi: single;
begin
  //        ska(N, R0) + d
  //   R0 -  --------------     * RD      //RD lässt sich scheinbar nicht wegkürzen...
  //           ska(N, RD)

  result := plusvector(ray.Origin, malvector(malvector(Ray.Direction, skalar(PlaneNormal,
Ray.Origin) + planeD) ,
                                      1/skalar(PlaneNormal, Ray.Direction)));
end;

function Col_Plane_Point(const Normal, Point: TVector3f; planeD: single): boolean;
var
  L: single;
begin
  L := PlaneD - skalar(normal, Point);
  result := abs(L) < 0.000001;
end;

//Achtung, ist nicht genau. aber für gewisse Zwecke ausreichend.
function Col_Triangle_Sphere(const Triangle: PPolygon; const Sphere: TSpheref): boolean;
var
  P: TVector3f;
  L: single;
begin
  result := false;
  //Ebene prüfen
  L := Triangle.d - skalar(Triangle.Vertexes[0].normal, Sphere.center);
  if abs(L) < Sphere.radius then
  begin
    //kreuzungspunkt berechnen
    P := PlusVector(Sphere.Center, MalVector(Triangle.Vertexes[0].normal,L));
    result := Col_Triangle_Point(Triangle,P);
  end;
end;

function Col_Triangle_Point(const Triangle: PPolygon; const Point: TVector3f): boolean;
var
  i, i2, counter: integer;
```

```
  Lx: single;  // koordinate des Schnittpunktes (x)
  x1,x2,y1,y2,px,py: single;  //Triangle Punkte
//  TakePlane2: boolean;
begin
  counter := 0;
  py := Point.y;
  px := Point.x;
  for i := 0 to 2 do
  begin
    i2 := (i+1) mod 3;
    with Triangle^ do
    begin
//      if (Vertexes[0].normal.y <> 0) or (Vertexes[0].normal.z<>0) then
//      if (abs(Vertexes[0].normal.x) <> 1)  then
      if (abs(Vertexes[0].normal.z)>0.1) then
//      if TakePlane2 then
      begin           // Teststrahl in + x-Richtung
        x1 := Vertexes[i].vertex.x;
        x2 := Vertexes[i2].vertex.x;
        y1 := Vertexes[i].vertex.y;
        y2 := Vertexes[i2].vertex.y;
      end else
        if (abs(Vertexes[0].normal.x) > 0.1) then
        begin  // Teststrahl in + z-Richtung (z-Koord werden in x1, x2 kopiert)
          x1 := Vertexes[i].vertex.z;
          x2 := Vertexes[i2].vertex.z;
          y1 := Vertexes[i].vertex.y;
          y2 := Vertexes[i2].vertex.y;
          px := Point.z;
        end else
        begin  // Teststrahl in + y-Richtung (z-Koord werden in y1, y2 kopiert)
          x1 := Vertexes[i].vertex.x;
          x2 := Vertexes[i2].vertex.x;
          y1 := Vertexes[i].vertex.z;
          y2 := Vertexes[i2].vertex.z;
          py := Point.z;
        end;

      if (y1-py)*(y2-py) < 0 then

        if ((x1-px)>0) and ((x2-px)>0) then
          inc(counter)
        else
          begin
            Lx := ((x2*y1)-(x1*y2)+(py*(x1-x2)))/(y1-y2);
            if Lx-px > 0 then
              inc(counter);
          end;
    end;
  end;
    result := (counter = 1);
end;
```

```
function Col_Triangle_Ray_Return_Dist(const Triangle: PPolygon; const Ray: TRayf; var Dist:
single): boolean;
var
  t: double;
  P: TVector3f;
  divi: single;
begin
  //see: http://www.devmaster.net/wiki/Ray-triangle_intersection
  //P = (1 - u - v)A + uB + vC          //Barycentric coordinates
  //Aber die Triangle_Point Methode scheint schneller zu sein.
  //dist = ((Ray.Origin-Triangle.Vertex[0])*Normal) / (Ray.Direction*Normal)   -> kürzen
  divi := Skalar(Ray.Direction, Triangle.Vertexes[0].normal);
  if abs(divi) < 0.000001 then
  begin
    result := false;
    Exit;
  end;
  dist := (Triangle.d-Skalar(Ray.Origin,Triangle.Vertexes[0].normal)) / divi;

  P := PlusVector(Ray.Origin, MalVector(Ray.Direction, dist));        //Punkt berechnen
  result := Col_Triangle_Point(Triangle, P);
end;

function Col_Triangle_Ray(const Triangle: PPolygon; const Ray: TRayf; Maxdist: single): boolean;
var
  t: double;
  dist: single;
  P: TVector3f;
  divi: single;
begin
  //see: http://www.devmaster.net/wiki/Ray-triangle_intersection
  //P = (1 - u - v)A + uB + vC          //Barycentric coordinates
  //Aber die Triangle_Point Methode scheint schneller zu sein.
  //dist = ((Ray.Origin-Triangle.Vertex[0])*Normal) / (Ray.Direction*Normal)   -> kürzen
  divi := Skalar(Ray.Direction, Triangle.Vertexes[0].normal);
  if abs(divi) < 0.000001 then
  begin
    result := false;
  end;
  dist := (Triangle.d-Skalar(Ray.Origin,Triangle.Vertexes[0].normal)) / divi;
  if (dist > Maxdist) or (dist < 0) then
  begin
    result := false;
    Exit;
  end;
  P := PlusVector(Ray.Origin, MalVector(Ray.Direction, dist));        //Punkt berechnen
  result := Col_Triangle_Point(Triangle, P);
end;

function Col_Sphere_Ray(const Sphere: TSpheref; const Ray: TRayf): boolean;
var
```

```
  t: double;
  //nearpoint, tempvector: TVector3f;
  tempv, Distv: TVector3f;
begin
  Distv := minusVector(Sphere.center, Ray.Origin);
  t := skalar(Ray.Direction, Distv);
  if t < 0 then
    t := 0;
  tempv := minusvector(malVector(Ray.Direction, t), Distv);
  //nearpoint:= plusVector(Ray.Origin, malVector(Ray.Direction, t));  //nächsten punkt berechnen
  //tempvector:= minusvector(nearpoint,Sphere.center);
  if (skalar(tempv, tempv)) < (Sphere.radius*Sphere.radius) then
    result := true
  else
    result := false;
end;

//------------------------------------------------------------------------------
//  OpenGL
//------------------------------------------------------------------------------
function CreateEmptyTexture(Width, Height, bpp: integer): Cardinal;
var
  pTexData: Pointer;
begin
  glEnable(GL_TEXTURE_2D);
  GetMem(pTexData, Width*Height*bpp);

  // Textur generieren und drauf zeigen
  glGenTextures(1, @result);
  glBindTexture(GL_TEXTURE_2D, result);

  // Daten in den Speicher, Lineares Filtering aktivieren
  glTexImage2D(GL_TEXTURE_2D, 0, bpp, Width, Height, 0, GL_RGB, GL_UNSIGNED_BYTE,
pTexData);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

  // Freigeben
  FreeMem(pTexData);
end;


//------------------------------------------------------------------------------
//  Misc
//------------------------------------------------------------------------------

function isInteger(s: String): boolean;
begin
  try
    Strtoint(s);
    result := true;
  except
```

```
    on Exception : EConvertError do
      result := false;
  end;
end;

function isFloat(s: String): boolean;
begin
  try
    Strtofloat(s);
    result := true;
  except
    on Exception : EConvertError do
      result := false;
  end;
end;

function VectorToString(const Vec: TVector3f): string;
begin
  result := floattostr(Vec.x) + '|' + floattostr(Vec.y) + '|' + floattostr(Vec.z);
end;

function PolygonToString(Poly: PPolygon): string;
begin
  result := 'Points: ' + VectorToString(Poly.Vertexes[0].Vertex) + ' - ' +
VectorToString(Poly.Vertexes[1].Vertex) +
          ' - ' + VectorToString(Poly.Vertexes[2].Vertex) + ' D: ' + floattostr(Poly.D) +
          ' Normal: ' + VectorToString(Poly.Vertexes[0].Normal);
end;

end.
```

4. CamUnit

```
unit CamUnit;
{
history:
 06.01.07: Ich habe nun angefangen die Kamera-Unit komplett zu überarbeiten.



}
interface

uses
  dglopenGL,
  Basics,
  UCharacters;

type
{  Tcamobject = record
    eye:              TVector;
    center:           TVector;
```

```
    up:             TVector;
    angley:          double;
  end;  }

  TCam = class(TObject)
    rotate: TVector2f;  //die Mausposition.
    procedure setcam(Char: TCharacter);      //set the cam with player's anglex and angley
    procedure Init(OwnChar: TCharacter);
    procedure setCamPos(Char: TCharacter);   //set the anglex and angley with rotate vector
  end;

var
  Cam: TCam;

implementation

uses
  Mainunit;

procedure TCam.setcam(Char: TCharacter);
begin
  glMatrixMode(GL_PROJECTION);
  //glLoadIdentity;

  with Char.Position do
  begin
    case game.gameVAR.cammode of
      0: begin
           glRotatef(lookangley,1,0,0);       //must be first
           glRotatef(lookanglex+90,0,1,0);      //+90 is important because OpenGL has another
Identity.
           glTranslatef(-X-sin(lookanglex*DEGTORAD)*0.2,-Char.HeadSphere.center.y,-
Z+cos(lookanglex*DEGTORAD)*0.2);
         end;

      1: begin
           glRotatef(lookangley,1,0,0);
           glRotatef(lookanglex+90,0,1,0);
           glTranslatef(-X,-Char.HeadSphere.center.y,-Z);
           glTranslatef(0.5, 0, 0.5);
         end;

      2: begin
           glRotatef(lookangley,1,0,0);
           glRotatef(lookanglex+90,0,1,0);
           glTranslatef(-X,-Char.HeadSphere.center.y,-Z);
           glTranslatef(0, -5, 0);
         end;

      3: begin
           glTranslatef(2,0,0);
           glRotatef(lookangley,1,0,0);       //must be first
```

```
        glRotatef(lookanglex+90,0,1,0);      //+90 is important because OpenGL has another
Identity.
        //glRotatef(lookangley,1,0,0);       //must be first
        //glRotatef(lookanglex+90,0,1,0);      //+90 is important because OpenGL has another
Identity.
        glTranslatef(-X,-Char.HeadSphere.center.y,-Z);
       end;
   end;
  end;
  glMatrixMode(GL_MODELVIEW);
end;

procedure TCam.Init(OwnChar: TCharacter);
begin
  rotate.x := 0;
  rotate.y := 0;
  setCamPos(OwnChar);
end;

procedure TCam.setCamPos(Char: TCharacter);
begin
  //if rotate.y < 100 then rotate.y := 100;
  //if rotate.y > 170 then rotate.y := 170;
  with Char.Position do
  begin
    Char.Position.lookanglex := rotate.x * gameVariabeln.MouseConfig.MouseSpeed;
    Char.Position.lookangley := rotate.y * gameVariabeln.MouseConfig.MouseSpeed;
  end;
end;

initialization
  cam := TCam.Create;

finalization
  cam.Free;




(*{-------------------------------------------------------------------------------}
{               setzt die Kamera an ein gewisses ort                    }
{-------------------------------------------------------------------------------}
procedure Tcam.setcam;
begin

  game.camera.eye.x      :=  game.chars[0].position.x;
  game.camera.eye.y      :=  game.chars[0].position.y + game.gameVAR.headheight;
  game.camera.eye.z      :=  game.chars[0].position.z;
{  game.camera.center.x   :=  game.chars[0].position.x + cos(mousex /
game.gameVar.mousesensivity * 2 * Pi);
  game.camera.center.y   :=  game.chars[0].position.y + sin(mousex /
```

```
game.gameVar.mousesensivity * 2 * Pi);
  game.camera.center.z   :=  game.chars[0].position.z + cos(mousey /
game.gameVar.mousesensivity * 2 * Pi)+ game.gameVAR.headheight;
}
  game.camera.center.x   :=  game.chars[0].position.x + cos(game.chars[0].position.angle);
  game.camera.center.y   :=  game.chars[0].position.y + cos(game.camera.angley) +
game.gameVAR.headheight;
  game.camera.center.z   :=  game.chars[0].position.z + sin(game.chars[0].position.angle);

  case game.gameVAR.cammode of

    0:       glulookat(  game.camera.eye.x,     game.camera.eye.y,     game.camera.eye.z,
                         game.camera.center.x,  game.camera.center.y,  game.camera.center.z,
                         game.camera.up.x,      game.camera.up.y,      game.camera.up.z            );

    1:       glulookat(  game.camera.center.x,  game.camera.center.y,  game.camera.center.z,
                         game.camera.eye.x,     game.camera.eye.y,     game.camera.eye.z,
                         game.camera.up.x,      game.camera.up.y,      game.camera.up.z            );

    2:       glulookat(  game.camera.center.x+5,  game.camera.center.y,
game.camera.center.z+5,
                         game.camera.eye.x,     game.camera.eye.y,     game.camera.eye.z,
                         game.camera.up.x,      game.camera.up.y,      game.camera.up.z            );
  end;


end;

procedure Tcam.Start;
begin
  game.camera.eye.x     :=  1;
  game.camera.eye.y     :=  1;
  game.camera.eye.z     :=  1;
  game.camera.center.x   :=  1;
  game.camera.center.y   :=  1;
  game.camera.center.z   :=  1;
  game.camera.up.x      :=  0;
  game.camera.up.y      :=  1;
  game.camera.up.z      :=  0;
end;    *)

end.
```

5. Eventhandler

```
unit EventHandlerUnit;

interface

uses
  Windows,
  SDL,
```

```
  ULogger,
  SysUtils,
  SDL_Net,
  classes,
  SyncObjs,
  UNetwork,
  Basics,
  Math,
  UScreenShot;

type
  TEventHandler = class(TObject)
    counter: Cardinal;

    lastkeydown: Cardinal;
    lastmousedown: Cardinal;
    leftmouseisDown: boolean;
    keyisDown:  boolean;

    //timebased movement; last changes
    lastrepetition:     Cardinal;
    lastcursorchange:   Cardinal;
    lastNetSend:        Cardinal;
    lastNetInfo:        Cardinal;
    last3Second:        Cardinal;
    lastReload:         Cardinal;
    constructor Create;
    destructor Destroy; override;
    procedure TimebasedMovement;
    procedure KeyDown(keysym : PSDL_keysym);
    procedure KeyUp(keysym : PSDL_keysym);
    procedure Mousemotion(motion: PSDL_MouseMotionEvent);
    procedure MouseUp(button: PSDL_MouseButtonEvent);
    procedure MouseDown(button: PSDL_MouseButtonEvent);
    procedure Resize(newWidth, newHeight: integer);
  end;

  TThreadPacket = record
    socket: PTCPSocket;
    data:   TTCPPackage;
    len:    integer;
  end;

  TPacketarray = array [0..30] of TThreadPacket;

  TPacketCollector = class(TObject)
  private
    processed:      boolean;
    currentPacket:  integer;
    InPackets:      TPacketArray;
    Outpackets:     TPacketArray;
    CritSection1:   TCriticalSection;
```

```
  public
    procedure AddPacket(socket: PTCPSocket; pack: TTCPPackage; len: integer);
    procedure ProcessPackets;
  end;

  TPacketThread = class(TThread)
  protected
    procedure Execute; override;
  end;

var
  PackCollector: TPacketCollector;
  PackThread:    TPacketThread;

implementation

uses
  Mainunit, Unit_SDL, GUI, CamUnit, GUIAdds, UCharacters, UShader, MovementUnit;

constructor TEventHandler.Create;
begin
  inherited Create;
  PackCollector := TPacketCollector.Create;
  PackCollector.CritSection1 := TCriticalSection.Create;
  PackCollector.currentPacket := 0;

  PackThread    := TPacketThread.Create(true);
  PackThread.FreeOnTerminate := True;
  PackThread.Resume;
  //Timer := SDL_Addtimer(10, @TEventHandler.Timer_1ms, nil);
end;

destructor TEventHandler.Destroy;
begin
  //SDL_RemoveTimer(Timer);
  inherited Destroy;
end;

procedure TEventHandler.TimebasedMovement;
var
  i: integer;
begin
  counter := SDL_GetTicks;
  //schüsse wiederholen
  if leftmouseisdown then
    if not game.showcursor then
      if not chars.CurrentChar.Dead and (Chars.CurrentChar.currentweaponslot = 0) then
        Chars.CurrentChar.HitLeftMouse;

  //Wiederholung des letzten Buchstabens
  if ((counter - lastmousedown) > 500) and keyisDown and
     (counter - lastrepetition >= 60) then
```

```
  begin
    lastrepetition := counter;
    GUIclass.KeyDown(GUIclass.lastkeydown);
  end;

  //GUIcursor
  if (counter - lastCursorChange >= 500) then
  begin
    GUIclass.Showcursor := not GUIclass.Showcursor;
    lastCursorChange := counter;
  end;

  if counter - lastNetSend >= 40 then
  begin
    Net.SendPlayerData;
    lastNetSend := counter;
  end;

  if (counter - last3Second >= 3000) then
  begin
    if Net.IsServer then
    begin
      //Sende Informationen...
      for i := 0 to Net.Server.numClients - 1 do
        if Net.Server.Clients[i] <> nil then
          Net.Server.Clients[i].SendTCPPacket(nServer_HealthChange,
inttostr(Chars.char[i+1].health), '', '');

      //Berechne die Spielerzahlen
      Chars.CalcLivingPlayers;
    end;

    if game.ingame then
    //Berechne die Statistiken
      GUIAdd.CalcStats;

    last3Second := counter;

  end;

  if counter - lastNetInfo >= 500 then
  begin
    if Net.IsClient then
    begin
      if Net.Client.hasInfos then
      begin
        Net.Client.SendInfos;
        Net.Client.hasInfos := false;
      end;
    end
    else
      if Net.IsServer then
```

```
    begin
      for i := 0 to Net.Server.numClients - 1 do
        if Net.Server.Clients[i] <> nil then
          if Net.Server.Clients[i].hasInfos then
          begin
            Net.Server.SendInfosToAll(i+1);
            Net.Server.Clients[i].hasInfos := false;
          end;
      if Net.Server.hasInfos then
      begin
        Net.Server.SendInfosToAll(0);
        Net.Server.hasInfos := false;
      end;
    end;
  end;

  if game.ingame then
  begin
    //Respawn
    with Chars do
    if (counter - NextRespawn >= 0) and NewRespawn then
    begin
      NewRespawn := false;
      for i := 0 to numChars - 1 do
      begin
        char[i].health     := 100;
        char[i].MoveSphere(game.CurrentLevel.GetRespawnPos(Char[i].Terrorist));
        if Length(Net.Server.Clients) > 0 then
          if (Net.Server.Clients[i - 1] <> nil) and (i > 0) then
            if (Net.Server.Clients[i - 1].ingame) then
            begin
              Net.Server.Clients[i - 1].SendTCPPacket(nServer_HealthChange, inttostr(100), '', '');
              Net.Server.Clients[i - 1].SendPosition(vec3f(char[i].position.x, char[i].position.y,
char[i].position.z));
              if char[i].dead then
              begin
                Net.Server.Clients[i - 1].SendTCPPacket(nServer_ArmorChange, inttostr(0), '', '');
                char[i].dead  := false;
              end;
            end;
      end;
    end;
  end;

end;

procedure TEventHandler.KeyDown(keysym : PSDL_keysym);
var
  temppoly : PPolygon;
  i: integer;
begin
  lastMouseDown := Counter;
```

```
  keyisDown     := true;
  GUIclass.KeyDown(keysym.sym);
  {case keysym.sym of
    //game.gameVar.keys.backw      :   game.gameVar.keys.backw.state    :=     true;
    SDLK_ESCAPE : done := -1;
  end; }
  //else if keysym.sym = game.gameVar.keys.fullscreen.which then begin
SDL_WM_ToggleFullScreen(surface); SDL_SetVideoMode( Screen_Width, Screen_Height, 32,
videoflags ); end
      if keysym.sym = game.gameVar.keys.exit.which       then begin
                                                  game.gameVar.keys.exit.state      :=     true;
                                                  if game.ingame and game.showcursor then
                                                  begin
                                                    game.showcursor := false;
                                                    for i := 0 to
Length(GUIclass.Windows[wIngame].ChildWindows) - 1 do
                                                      GUIclass.Windows[GUIclass.Windows[wIngame].C
hildWindows[i].Index].Visible := false;
                                                  end else
                                                    if game.runninggame then
                                                      game.ingame := not game.ingame;
                                                end
  else if keysym.sym = SDLK_BACKQUOTE                    then
                                               begin
                                                 if not Guiclass.Windows[Console.Window].Visible then
                                                 begin
                                                   game.ingame := false;
                                                   GUIclass.Windows[Console.Window].Visible := true;
                                                   GUIclass.lastMouseUp := Console.Edit;
                                                 end
                                                 else
                                                   if game.runninggame then
                                                     game.ingame := true
                                                   else
                                                     Guiclass.Windows[Console.Window].Visible := false;
                                               end
  else if keysym.sym = SDLK_PRINT                        then
                                               begin
                                                 i := 0;
                                                 while FileExists('Screenshots\Screen' + inttostr(i) +
'.jpg') do
                                                   inc(i);
                                                 ScreenShot('Screenshots\Screen' + inttostr(i) + '.jpg');
                                               end
  else if game.ingame then
  begin
       if keysym.sym = game.gameVar.keys.backw.which      then
game.gameVar.keys.backw.state       :=     true
   else if keysym.sym = game.gameVar.keys.forw.which       then game.gameVar.keys.forw.state
:=     true
   else if keysym.sym = game.gameVar.keys.left.which       then game.gameVar.keys.left.state
:=     true
```

```
   else if keysym.sym = game.gameVar.keys.right.which      then game.gameVar.keys.right.state
:=     true
   else if keysym.sym = game.gameVar.keys.camchange.which  then game.gameVar.cammode
:=     (game.gameVar.cammode + 1) mod game.gameVar.cams
   else if keysym.sym = game.gameVar.keys.pause.which      then
game.gameVar.keys.pause.state       :=     true
   else if keysym.sym = SDLK_1                             then chars.CurrentChar.currentweaponslot :=
0
   else if keysym.sym = SDLK_2                             then chars.CurrentChar.currentweaponslot :=
1
   else if keysym.sym = SDLK_3                             then chars.CurrentChar.currentweaponslot :=
2
   else if keysym.sym = SDLK_4                             then chars.CurrentChar.currentweaponslot :=
3
   else if keysym.sym = SDLK_TAB                            then GUIclass.Windows[wStatistik].Visible
:= true
   else if keysym.sym = SDLK_R                             then Chars.CurrentChar.Reload
   else if keysym.sym = game.gameVar.keys.jump.which       then
                                          begin
                                            if abs(Movement.CurrentYSpeed) < 0.001 then
                                            begin      // v(y -> ymax) = sqrt(y*2g)
                                              Movement.CurrentYSpeed := 4;  //y=0.8m
                                            end;
                                          end
   else if keysym.sym = SDLK_M                              then
                                          begin
                                             GUIclass.Windows[wChars].Visible := true;
                                             game.ShowCursor := true;
                                          end
   else if keysym.sym = SDLK_B                             then
                                          begin
                                             GUIclass.Windows[wBuy].Visible := not
GUIclass.Windows[wBuy].Visible;
                                             if GUIclass.Windows[wBuy].Visible then
                                               game.ShowCursor := true
                                             else
                                               game.ShowCursor := false;
                                          end
   else if keysym.sym = SDLK_C                             then
                                          begin
                                             if not GUIclass.Windows[wChat].Visible then
                                             begin
                                               GUIclass.Windows[wChat].Visible := true;
                                               game.ShowCursor := true;
                                               GUIclass.Windows[wChat].X := halfscreenWidth  -
(GUIclass.Windows[wChat].Width  div 2);
                                               GUIclass.Windows[wChat].Y := halfscreenHeight -
(GUIclass.Windows[wChat].Height div 2);
                                               GUIclass.Windows[wChat].Edits[0].Text := '';
                                               GUIclass.Windows[wChat].CloseButton.Visible :=
true;
                                             end;
```

```
                                        end
  else if keysym.sym = SDLK_F11                then
                                        begin
                                          temppoly :=
game.CurrentLevel.GetTriangleData(Chars.CurrentChar.GetRay);
                                          if temppoly <> nil then
                                          begin
                                            Console.AddStringAndLog('Pick :: ' +
PolygonToString(temppoly), 'EventHandler');
                                            temppoly.material := 0;
                                          end;
                                        end
  else if keysym.sym = SDLK_PRINT                then
                                        begin
                                          i := 0;
                                          while FileExists('Screenshots\Screen' + inttostr(i) +
'.jpg') do
                                            inc(i);
                                          ScreenShot('Screenshots\Screen' + inttostr(i) + '.jpg');
                                        end;
 end;

end;

procedure TEventHandler.KeyUp(keysym : PSDL_keysym);
begin
 keyisdown := false;
 { case keysym.sym of
   SDLK_ESCAPE : done := -1;
 end; }
     if keysym.sym = game.gameVar.keys.backw.which     then game.gameVar.keys.backw.state
:=    false
 else if keysym.sym = game.gameVar.keys.forw.which     then game.gameVar.keys.forw.state
:=    false
 else if keysym.sym = game.gameVar.keys.left.which     then game.gameVar.keys.left.state
:=    false
 else if keysym.sym = game.gameVar.keys.right.which    then game.gameVar.keys.right.state
:=    false
 else if keysym.sym = game.gameVar.keys.jump.which     then game.gameVar.keys.jump.state
:=    false
 else if keysym.sym = game.gameVar.keys.pause.which    then game.gameVar.keys.pause.state
:=    false
 else if keysym.sym = game.gameVar.keys.exit.which     then game.gameVar.keys.exit.state
:=    false
 else if keysym.sym = SDLK_TAB                    then GUIclass.Windows[wStatistik].Visible:=
false;
 GUIclass.KeyUp(keysym.sym);
end;

procedure TEventHandler.Mousemotion(motion: PSDL_MouseMotionEvent);
var
 tempmotion: integer;
```

```
begin
  {game.chars[0].position.angle:=(game.chars[0].position.angle*game.gameVar.mousesensivity/
2/Pi + motion.x - halfscreenwidth ) / game.gameVar.mousesensivity*2*Pi;
  game.camera.angley:=          (game.camera.angley*game.gameVar.mousesensivity/2/Pi +
motion.y - halfscreenheight) / game.gameVar.mousesensivity*2*Pi;
  if game.camera.angley > Pi then game.camera.angley := Pi;
  if game.camera.angley < 0  then game.camera.angley := 0;  }
  if Game.ingame and not game.ShowCursor then
  begin
  {  tempmotion := (motion.x - halfscreenwidth);
    if tempmotion < 0 then
      Cam.rotate.x := Cam.rotate.x - power(tempmotion*gameVariabeln.MouseConfig.MouseSpeed,
                                  gameVariabeln.MouseConfig.MouseAcc)
    else
      Cam.rotate.x := Cam.rotate.x +
power(tempmotion*gameVariabeln.MouseConfig.MouseSpeed,
                                  gameVariabeln.MouseConfig.MouseAcc);

    tempmotion := (motion.y - halfscreenheight);
    if tempmotion < 0 then
      Cam.rotate.y := Cam.rotate.y - power(tempmotion*gameVariabeln.MouseConfig.MouseSpeed,
                                   gameVariabeln.MouseConfig.MouseAcc)
    else
      Cam.rotate.y := Cam.rotate.y +
power(tempmotion*gameVariabeln.MouseConfig.MouseSpeed,
                                   gameVariabeln.MouseConfig.MouseAcc);       }

      Cam.rotate.x := Cam.rotate.x + ((motion.x - halfscreenwidth )
*gameVariabeln.MouseConfig.MouseSpeed);
      Cam.rotate.y := Cam.rotate.y + ((motion.y -
halfscreenheight)*gameVariabeln.MouseConfig.MouseSpeed);


    Cam.setCamPos(Chars.CurrentChar);
  end;
  GUIclass.MouseMove(motion.x,motion.y);
end;

{
    Button 1:          Left mouse button
    Button 2:          Middle mouse button
    Button 3:          Right mouse button
    Button 4:          Mouse Wheel Up
    Button 5:          Mouse Wheel Down
}

procedure TEventHandler.MouseDown(button: PSDL_MouseButtonEvent);
begin
  if gameVariabeln.MouseConfig.InvertMouse then
    case button.button of
      1: button.button := 3;
      3: button.button := 1;
```

```
    end;
  lastMouseDown := Counter;
  case button.button of
    1:  begin
         if not game.ShowCursor then
           if not chars.CurrentChar.Dead then
           begin
             Chars.CurrentChar.HitLeftMouse;
             leftmouseisdown := true;
           end;
        end;
    //2:  frei
    //3:  zoom
    4:  begin                    //needs improvement..
         if game.ingame then
           chars.CurrentChar.currentweaponslot := (chars.CurrentChar.currentweaponslot + 1) mod
4;
         //if game.weapons.currentweaponslot = 0 then
        end;
    5:  begin
         if game.ingame then
           chars.CurrentChar.currentweaponslot := (chars.CurrentChar.currentweaponslot - 1) mod
4;
        end;
  end;
  GUIclass.MouseDown(Button.x,button.y,button.button);
end;


procedure TEventHandler.MouseUp(button: PSDL_MouseButtonEvent);
begin
  if gameVariabeln.MouseConfig.InvertMouse then
    case button.button of
      1: button.button := 3;
      3: button.button := 1;
    end;
  case button.button of
    1:  leftMouseisdown := false;
    //2:  frei
    //3:  zoom weg
    //4:  freilassen!
    //5:  freilassen!
  end;
  GUIclass.MouseUp(Button.x,button.y,button.button);
end;


procedure TEventHandler.Resize(newWidth, newHeight: integer);
begin
  GUI.screen_width := newWidth;
  GUI.screen_height := newHeight;
  MainUnit.Screen_width := newWidth;
```

```
  MainUnit.Screen_height := newHeight;
  MainUnit.halfscreenwidth := newWidth div 2;
  MainUnit.halfscreenheight := newHeight div 2;
  if game.LoadedMenus then
  begin
    GUIAdd.Resize;
    Renderpass1.onScreenResize(Screen_Width, Screen_Height, false);
  end;
end;

procedure TPacketThread.Execute;
begin
  while not Terminated do
  begin
    try
      if not PackCollector.processed then
        PackCollector.ProcessPackets
      else
        sleep(20);
    except
      on e: exception do begin
        game.onError('Problem mit TPacketThread', 'Eventhandler');
      end;
    end;
  end;
end;

procedure TPacketCollector.AddPacket(socket: PTCPSocket; pack: TTCPPackage; len: integer);
begin
  //CritSection1.Enter;
  Log.AddWarning('Added a Packet (TPacketCollector): ' +
Net_MessageTypeToStr(pack.MessageType), 'Eventhandler');
  processed := false;
  InPackets[currentPacket].socket := socket;
  InPackets[currentPacket].data   := pack;
  InPackets[currentPacket].len    := len;
  inc(currentPacket);
  //CritSection1.Leave;
end;

procedure TPacketCollector.ProcessPackets;
var
  i: integer;
begin
  //CritSection1.Enter;
  for i := 0 to CurrentPacket - 1 do
    OutPackets[i] := InPackets[i];
  //CritSection1.Leave;
  for i := 0 to CurrentPacket - 1 do
  begin
    SDLNet_TCP_Send(OutPackets[i].socket, @OutPackets[i].data, OutPackets[i].len);
    Log.AddWarning('Sent a Packet (TPacketCollector): ' +
```

```
Net_MessageTypeToStr(OutPackets[i].data.MessageType), 'Eventhandler');
  end;
  currentPacket := 0;
  processed := true;
end;

end.
```

6. Filetypes

```
unit FileTypes;

{  Filetype Unit für das abspeichern von verschiedenen Dateien,
   es wird schlussendlich eine Text und Binary Version geben...

  start des programmierens: 7.11.06


  Die Strings in TDTFile sind Dollar-terminiert! Das heisst, man darf nicht mit Dollars schreiben...
}

interface

uses
  classes, dialogs, sysutils, Variants;
const
  VERSION = '0.1';
  DEVELOPER ='David Halter';

type
  TVariantarray = array of Variant;

  PLine = ^TLine;
  TLine = record
    wordNr:    integer;
    lineNr:    integer;
    strings:   array of string;
  end;

  //Text
  TDTFile = class(TObject)  {Dollar - Text - File}
  private
    _CrntLine:  TLine;          //Current Line...
    Txtfile:    TextFile;
    _developer: string;
    _typ:       string;
    _version:   string;
    _write:     boolean;
    procedure ReadHeader;

  public
    constructor Create(Path: string; write: boolean; add: boolean = false);
    procedure   CloseFile;
```

```
  //Read
  function  Readln: PLine;   overload;
  procedure Readln(var Line: TLine);   overload;
  procedure Readln(var Data: array of const); overload;
  function  Read: String;

  //Write
  procedure Writeln(PLine: PLine); overload;
  procedure Writeln(const TLine: TLine); overload;
  procedure Writeln(const Data: array of const); overload;
  procedure Write(s: string);
  procedure WriteHeader(version: string; typ: string);

  property  CrntLine:  TLine   read _CrntLine;    //current line...
  property  developer: string  read _developer;
  property  typ: string        read _typ;
  property  version: string    read _version;
end;

TBinaryFile = class(TObject)
private
  _writefile: boolean;
  _path:      string;
  _developer: string;
  _typ:       string;
  _version:   string;
public
  S:  TFileStream;
  constructor Create(Path: string; write: boolean; add: boolean = false);
  procedure CloseFile;
  //procedure Write(const Data);                  overload;
  procedure Write(const Data: TVariantarray);     overload;
  procedure Write(const Data: String);           overload;
  procedure Write(const Data: Word);              overload;
  procedure Write(const Data: Single);           overload;
  procedure Write(const Data: Double);            overload;
  procedure Write(const Data: Integer);          overload;
  procedure Write(const Data: Smallint);          overload;
  procedure Write(const Data: Boolean);           overload;
  procedure Write(const Data: Cardinal);          overload;
  procedure Write(const Data: array of const);    overload;

  //procedure Read(var Data);                     overload;
  procedure Read(var Data: TVariantarray);        overload;
  procedure Read(var Data: String);              overload;
  procedure Read(var Data: Word);                 overload;
  procedure Read(var Data: Single);              overload;
  procedure Read(var Data: Double);               overload;
  procedure Read(var Data: Integer);             overload;
  procedure Read(var Data: Smallint);             overload;
  procedure Read(var Data: Boolean);              overload;
```

```
    procedure Read(var Data: Cardinal);           overload;
    procedure ReadHeader;
    procedure WriteHeader(version: string; typ: string);

    property  Path: string       read _path;
    property  writeFile: boolean read _writefile;
    property  developer: string  read _developer;
    property  typ: string        read _typ;
    property  version: string    read _version;
  end;

implementation

{-------------------------------------------------------------------}
{#################################################################
############}
{               DTF - DOLLAR TEXT FILE
{#################################################################
############}
{-------------------------------------------------------------------}

constructor TDTFile.Create(Path: string; write: boolean; add: boolean = false);
begin
  _CrntLine.lineNr := -1;
  _CrntLine.wordNr := -1;
  setlength(_CrntLine.strings,1);
  _CrntLine.strings[0] := '';
  AssignFile(TxtFile, Path);
  if write then Rewrite(TxtFile)
  else Reset(TxtFile);
  _write := write;
  ReadHeader;
end;

procedure TDTFile.CloseFile;
begin
  system.CloseFile(TxtFile);
  Free;
end;

{#################################################################
######}
{#######################  Read Anweisungen  ######################}
{#################################################################
######}
function  TDTFile.Readln: PLine;
var
  s:       string;
  i:       integer;
  a:       array of integer;
begin
  if _write then
```

```
    showmessage('Cannot Read into a Write - Only File');
  if EOF(TxtFile) then
    showmessage('End of file reached..');

  inc(_CrntLine.lineNr);
  _CrntLine.wordNr := -1;
  setlength(_crntline.strings, 0);
  System.Readln(TxtFile, s);

  for i:= 0 to length(s) - 1 do
    if s[i] = '$' then
    begin
      setlength(a,High(a)+2);
      a[High(a)] := i;
    end;

  for i := 0 to high(a) do
  begin
      setlength(_crntline.strings, High(_crntline.strings) + 2);
      if i = 0 then
      begin
        if a[0] = 1 then
          _crntline.strings[High(_crntline.strings)] := ''
        else
          _crntline.strings[High(_crntline.strings)] := copy(s,0,a[0]-1);
      end else
      begin
        if a[i]+1 = a[i+1] then
          _crntline.strings[High(_crntline.strings)] := ''
        else
          _crntline.strings[High(_crntline.strings)] := copy(s,a[i]+1,a[i+1]-a[i]-1);
      end;
    end;
  result := @crntline;
end;

procedure TDTFile.Readln(var Line: TLine);
begin
  Line := Readln^;
end;

procedure TDTFile.Readln(var Data: array of const);
var
  i: integer;
begin
  Readln;
  for i := 0 to High(Data) do
    if i<=high(_CrntLine.strings) then
      case Data[i].VType of
        vtInteger:     Data[i].VInteger          := StrToInt(_crntline.strings[i]);
        vtBoolean:     Data[i].VBoolean          := StrToBool(_crntline.strings[i]);
        vtChar:        Data[i].VChar             := PChar(_crntline.strings[i])^;
```

```
      vtExtended:    Data[i].VExtended^        := strtofloat(_crntline.strings[i]);
      vtString:      Data[i].VString^          := _crntline.strings[i];
      vtPChar:       Data[i].VPChar            := PChar(_crntline.strings[i]);
      //vtObject:      Data[i].VObject.ClassName  := _crntline.strings[i];
      //vtClass:       Data[i].VClass.ClassName   := _crntline.strings[i];
      vtAnsiString:  Data[i].VAnsiString       := @_crntline.strings[i];//string(Data[i].VAnsiString);
      vtCurrency:    Data[i].VCurrency^        :=
StrToCurr(_crntline.strings[i]);//CurrToStr(Data[i].VCurrency^);
      vtVariant:     Data[i].VVariant^         := string(Data[i].VVariant^);
      vtInt64:       Data[i].VInt64^           := StrToInt(_crntline.strings[i]);
    end;
end;

function  TDTFile.Read: String;
begin
  if (_CrntLine.wordNr = -1) or (_CrntLine.wordNr = High(_CrntLine.strings)) then
    Readln;
  inc(_CrntLine.wordNr);
  result := _CrntLine.strings[_CrntLine.wordNr];
end;

procedure TDTFile.ReadHeader;
begin
  Readln;
  _developer  :=  _CrntLine.strings[0];
  _typ        :=  _CrntLine.strings[1];
  _version    :=  _CrntLine.strings[2];
end;

{##########################################################################}
{######################   Write Anweisungen   ######################}
{##########################################################################}
procedure TDTFile.Writeln(PLine: PLine);
var
  i: integer;
  s: string;
begin
  if not _write then
    showmessage('Cannot Write into a Read - Only File');
  s := '';
  for i := 0 to High(PLine.strings) do
    s := s + PLine.strings[i] + '$';
  system.Writeln(TxtFile, s);
end;

procedure TDTFile.Writeln(const TLine: TLine);
begin
  Writeln(@TLine);
end;
```

```
procedure TDTFile.Writeln(const Data: array of const);
var
  i: integer;
begin
  setlength(_crntline.strings, High(Data));
  for i := 0 to High(Data) do
    case Data[i].VType of
      vtInteger:     _crntline.strings[i] := IntToStr(Data[i].VInteger);
      vtBoolean:     _crntline.strings[i] := BoolToStr(Data[i].VBoolean);
      vtChar:        _crntline.strings[i] := Data[i].VChar;
      vtExtended:    _crntline.strings[i] := FloatToStr(Data[i].VExtended^);
      vtString:      _crntline.strings[i] := Data[i].VString^;
      vtPChar:       _crntline.strings[i] := Data[i].VPChar;
      vtObject:      _crntline.strings[i] := Data[i].VObject.ClassName;
      vtClass:       _crntline.strings[i] := Data[i].VClass.ClassName;
      vtAnsiString:  _crntline.strings[i] := string(Data[i].VAnsiString);
      vtCurrency:    _crntline.strings[i] := CurrToStr(Data[i].VCurrency^);
      vtVariant:     _crntline.strings[i] := string(Data[i].VVariant^);
      vtInt64:       _crntline.strings[i] := IntToStr(Data[i].VInt64^);
    end;
  self.Writeln(@_crntline);
end;

procedure TDTFile.Write(s: string);
begin
  system.Write(TxtFile, s, '$');
end;

procedure TDTFile.WriteHeader(version: string; typ: string);
begin
  setlength(_crntline.strings, 3);
  _crntline.strings[0] := typ;
  _crntline.strings[1] := FileTypes.DEVELOPER;
  _crntline.strings[2] := version;
  Writeln(_crntline);
end;

{-------------------------------------------------------------------}
{#####################################################################
############}
{                BF - Binary FILE
{#####################################################################
############}
{-------------------------------------------------------------------}

constructor TBinaryFile.Create(Path: string; write: boolean; add: boolean = false);
begin
  _writefile := write;
  _path  := path;
  if add then
    S    := TFileStream.Create(path,fmCreate)
    else
```

```
    if write then
      S    := TFileStream.Create(path,fmCreate or fmOpenWrite)
    else
      S    := TFileStream.Create(path,fmOpenRead);
end;

procedure TBinaryFile.CloseFile;
begin
  S.Free;
  Free;
end;

{##########################################################################}
{#######################   Read Anweisungen   #######################}
{##########################################################################}
procedure TBinaryFile.ReadHeader;
begin
  read(_typ);
  read(_developer);
  read(_version);
end;

procedure TBinaryFile.Read(var Data: String);
var
  size: word;
begin
  S.Read(size, SizeOf(size));
  //wil not be called if size = 0
  if 0 <> size then
  begin
    SetLength(Data, size);
    S.Read(Pchar(Data)^, size);
  end
  else
    Data := '';
end;

procedure TBinaryFile.Read(var Data: Word);
begin
  S.Read(data, SizeOf(Word));
end;

procedure TBinaryFile.Read(var Data: Single);
begin
  S.Read(data, SizeOf(Single));
end;

procedure TBinaryFile.Read(var Data: Double);
begin
  S.Read(data, SizeOf(Double));
```

```
end;

procedure TBinaryFile.Read(var Data: Integer);
begin
  S.Read(data, SizeOf(Integer));
end;

procedure TBinaryFile.Read(var Data: Smallint);
begin
  S.Read(data, SizeOf(Smallint));
end;

procedure TBinaryFile.Read(var Data: Boolean);
begin
  S.Read(data, SizeOf(Boolean));
end;

procedure TBinaryFile.Read(var Data: Cardinal);
begin
  S.Read(data, SizeOf(Cardinal));
end;

{procedure TBinaryFile.Read(var Data);
begin
  S.Read(Data, SizeOf(Data));
end;                        }

procedure TBinaryFile.Read(var Data: TVariantarray);

var
  i:            integer;
  size:          word;
  //----types----
  _smallint:      smallint;
  _integer:       integer;
  _single:        single;
  _double:         double;
  _currency:       currency;
  _boolean:        boolean;
  _ShortInt:       ShortInt;
  _Byte:          Byte;
  _word:           word;
  _LongWord:      LongWord;
  _Int64:         Int64;
  _string:        string;
begin
  for i := 0 to High(Data) do
    case VarType(Data[i]) of
      varSmallint: {= $0002;  vt_i2          2 }
      begin
        S.Read(_Smallint,SizeOf(Smallint));
        Data[i] := _Smallint;
```

```
    end;

    varInteger:  {= $0003;  vt_i4         3 }
    begin
      S.Read(_integer,SizeOf(integer));
      Data[i] := _integer;
    end;

    varSingle:   {= $0004;  vt_r4         4 }
    begin
      S.Read(_single,SizeOf(single));
      Data[i] := _single;
    end;

    varDouble:   {= $0005;  vt_r8         5 }
    begin
      S.Read(_double,SizeOf(double));
      Data[i] := _double;
    end;

    varCurrency: {= $0006;  vt_cy         6 }
    begin
      S.Read(_currency,SizeOf(currency));
      Data[i] := _currency;
    end;

    varBoolean:  {= $000B;  vt_bool       11 }
    begin
      S.Read(_boolean,SizeOf(boolean));
      Data[i] := _boolean;
    end;

    varShortInt: {= $0010;  vt_i1         16 }
    begin
      S.Read(_ShortInt,SizeOf(ShortInt));
      Data[i] := _ShortInt;
    end;

    varByte:     {= $0011;  vt_ui1        17 }
    begin
      S.Read(_byte,SizeOf(byte));
      Data[i] := _byte;
    end;

    varWord:     {= $0012;  vt_ui2        18 }
    begin
      S.Read(_Word,SizeOf(word));
      Data[i] := _Word;
    end;

    varLongWord: {= $0013;  vt_ui4        19 }
    begin
```

```
      S.Read(_LongWord,SizeOf(Longword));
      Data[i] := _LongWord;
    end;

    varInt64:    {= $0014;  vt_i8        20 }
    begin
      S.Read(_Int64,SizeOf(Int64));
      Data[i] := _Int64;
    end;

    varString:   {= $0100;  Pascal string 256 } {not OLE compatible }
    begin
      S.Read(size,SizeOf(size));
      S.Read(Pchar(_string)^, size);
      Data[i] := _string;
    end;

    else
      S.Read(Data[i], SizeOf(Data[i]));
  end;

{var
  i: integer;
  size: word;
  str: string;
begin
  for i := 0 to High(Data) do
    if VarType(Data[i]) = varString then
    begin
      S.Read(size,SizeOf(size));
      S.Read(Pchar(str)^, size);
      Data[i] := str;
    end
    else
      S.Read(Data[i], SizeOf(Data[i])); }
end;

{##########################################################################}
{#####################   Write Anweisungen   ######################}
{##########################################################################}
procedure TBinaryFile.WriteHeader(version: string; typ: string);
begin
  Write(typ);
  Write(Filetypes.DEVELOPER);
  Write(version);
end;

{procedure TBinaryFile.Write(const Data);
begin
  S.Write(data, SizeOf(Data));
```

```
end;        }

procedure TBinaryFile.Write(const Data: Word);
begin
  S.Write(data, SizeOf(Word));
end;

procedure TBinaryFile.Write(const Data: Single);
begin
  S.Write(data, SizeOf(Single));
end;

procedure TBinaryFile.Write(const Data: Double);
begin
  S.Write(data, SizeOf(Double));
end;

procedure TBinaryFile.Write(const Data: Integer);
begin
  S.Write(data, SizeOf(Integer));
end;

procedure TBinaryFile.Write(const Data: Smallint);
begin
  S.Write(data, SizeOf(Smallint));
end;

procedure TBinaryFile.Write(const Data: Boolean);
begin
  S.Write(data, SizeOf(Boolean));
end;

procedure TBinaryFile.Write(const Data: Cardinal);
begin
  S.Write(data, SizeOf(Cardinal));
end;

procedure TBinaryFile.Write(const Data: String);
var
  size: word;
begin
  size := length(data);
  //This will not be called if Size = 0
  S.Write(size, Sizeof(word));
  if size <> 0 then
    S.Write(Pchar(Data)^, size);
end;

procedure TBinaryFile.Write(const Data: TVariantarray);
var
  i:          integer;
  size:         word;
```

```
  //----types----
  _smallint:      smallint;
  _integer:       integer;
  _single:        single;
  _double:        double;
  _currency:      currency;
  _boolean:       boolean;
  _ShortInt:      ShortInt;
  _Byte:          Byte;
  _word:          word;
  _LongWord:      LongWord;
  _Int64:         Int64;
  _string:        string;
begin
  for i := 0 to High(Data) do
    case VarType(Data[i]) of
      varSmallint: {= $0002;  vt_i2         2 }
      begin
        _Smallint := Data[i];
        S.Write(_Smallint,SizeOf(Smallint));
      end;

      varInteger:  {= $0003;  vt_i4         3 }
      begin
        _integer := Data[i];
        S.Write(_integer,SizeOf(integer));
      end;

      varSingle:   {= $0004;  vt_r4         4 }
      begin
        _single := Data[i];
        S.Write(_single,SizeOf(single));
      end;

      varDouble:   {= $0005;  vt_r8         5 }
      begin
        _double := Data[i];
        S.Write(_double,SizeOf(double));
      end;

      varCurrency: {= $0006;  vt_cy         6 }
      begin
        _currency:= Data[i];
        S.Write(_currency,SizeOf(currency));
      end;

      varBoolean:  {= $000B;  vt_bool       11 }
      begin
        _boolean := Data[i];
        S.Write(_boolean,SizeOf(boolean));
      end;
```

```
    varShortInt: {= $0010;  vt_i1        16 }
    begin
      _ShortInt := Data[i];
      S.Write(_ShortInt,SizeOf(ShortInt));
    end;

    varByte:    {= $0011;  vt_ui1       17 }
    begin
      _byte := Data[i];
      S.Write(_byte,SizeOf(byte));
    end;

    varWord:    {= $0012;  vt_ui2       18 }
    begin
      _Word := Data[i];
      S.Write(_Word,SizeOf(word));
    end;

    varLongWord: {= $0013;  vt_ui4       19 }
    begin
      _LongWord := Data[i];
      S.Write(_LongWord,SizeOf(Longword));
    end;

    varInt64:   {= $0014;  vt_i8        20 }
    begin
      _Int64 := Data[i];
      S.Write(_Int64,SizeOf(Int64));
    end;

    varString:   {= $0100;  Pascal string 256 } {not OLE compatible }
    begin
      _string := Data[i];
      size := length(_string);
      S.Write(size,SizeOf(size));
      S.Write(Pchar(_string)^, size);
    end;

    else
      S.Write(Data[i], SizeOf(Data[i]));
  end;

{   if VarType(Data[i]) = varString then
  begin
    str := Data[i];
    size := length(str);
    S.Write(size,SizeOf(size));
    S.Write(Pchar(str)^, size);
  end
  else
    S.Write(Data[i], SizeOf(Data[i]));    }
end;
```

```
procedure TBinaryFile.Write(const Data: array of const);
var
  i: integer;
  str: string;
  size: integer;
begin
  for i := 0 to High(Data) do
    if Data[i].VType = vtString then
    begin
      str := Data[i].vString^;
      size := length(str);
      S.Write(size,SizeOf(size));
      S.Write(Pchar(str)^, size);
    end
    else
      S.Write(Data[i], SizeOf(Data[i]));
end;

end.
```

7. GameVarUnit

```
unit gameVARUnit;

interface

uses
  Sysutils,
  Basics,
  IniFiles,
  SDL,
  DGLOpenGL,
  UShader;

type
  TFog = record
    FogColor:        TColor4f;
    FogStart:        double;
    FogEnd:          double;
    FogDensity:      double;
    FogMode:         integer;
    Enabled:         boolean;
  end;

  TgameKey = record
    which:           word;
    state:           boolean;
  end;

  Tkeys = record
    forw:            TgameKey;
    backw:           TgameKey;
```

```
    left:            TgameKey;
    right:           TgameKey;
    jump:            TgameKey;
    exit:            TgameKey;
    camchange:       TgameKey;
    pause:           TgameKey;
    fullscreen:      TgameKey;
  end;

  TAllConfig = class(TObject)
  private
    _iniFile: string;
    procedure LoadIndividual(Ini: TIniFile); virtual; abstract;
    procedure SaveIndividual(Ini: TIniFile); virtual; abstract;
  public
    constructor Create(__iniFile: string);
    destructor Destroy; override;
    procedure Load;
    procedure Save;

    property iniFile: string read _iniFile;
  end;

  TVideoConfig = class(TAllConfig)
  private
    _VerticalSync:    boolean;
    _Shader:          boolean;
    _WindowMode:      boolean;
    _Gamma:           single;
    _ResolutionX:     integer;
    _ResolutionY:     integer;
    _ResolutionIndex: integer;
    _GraphicQuality:  integer;
    _BitDepth:        integer;
    _BitDepthIndex:   integer;

    procedure _SetBitDepthIndex(Value: Integer);
    procedure _SetResolutionIndex(Value: Integer);
    procedure _SetShader(Value: boolean);
  public
    procedure LoadIndividual(Ini: TIniFile); override;
    procedure SaveIndividual(Ini: TIniFile); override;

    property VerticalSync:    boolean  read _VerticalSync     write _VerticalSync;
    property Shader:          boolean  read _Shader           write _SetShader;
    property WindowMode:      boolean  read _WindowMode       write _WindowMode;
    property Gamma:           single   read _Gamma            write _Gamma;
    property ResolutionX:     integer  read _ResolutionX      write _ResolutionX;
    property ResolutionY:     integer  read _ResolutionY      write _ResolutionY;
    property ResolutionIndex: integer  read _ResolutionIndex  write _SetResolutionIndex;
    property GraphicQuality:  integer  read _GraphicQuality   write _GraphicQuality;
    property BitDepthIndex:   integer  read _BitDepthIndex    write _SetBitDepthIndex;
```

```
  property BitDepth:        integer   read _BitDepth        write _BitDepth;
end;

TAudioConfig = class(TAllConfig)
private
  _Sound:          boolean;
  _Music:          boolean;
  _SoundVolume:     single;
  _MusicVolume:     single;
public
  procedure LoadIndividual(Ini: TIniFile); override;
  procedure SaveIndividual(Ini: TIniFile); override;

  property Sound:          boolean   read _Sound          write _Sound;
  property Music:          boolean   read _Music          write _Music;
  property SoundVolume:     single   read _SoundVolume     write _SoundVolume;
  property MusicVolume:     single   read _MusicVolume     write _MusicVolume;
end;

TKeyConfig = class(TAllConfig)
private

public
  procedure LoadIndividual(Ini: TIniFile); override;
  procedure SaveIndividual(Ini: TIniFile); override;


end;

TMouseConfig = class(TAllConfig)
private
  _MouseSpeed:      single;
  _MouseAcc:        single;
  _InvertMouse:     boolean;
public
  procedure LoadIndividual(Ini: TIniFile); override;
  procedure SaveIndividual(Ini: TIniFile); override;

  property MouseSpeed:      single   read _MouseSpeed      write _MouseSpeed;
  property MouseAcc:        single   read _MouseAcc        write _MouseAcc;
  property InvertMouse:     boolean  read _InvertMouse     write _InvertMouse;
end;

TMiscConfig = class(TAllConfig)
private
  _ShowDebug:       boolean;
  _Sleep:           integer;
  _NearClipping:    single;
  _FarClipping:     single;
public
  procedure LoadIndividual(Ini: TIniFile); override;
  procedure SaveIndividual(Ini: TIniFile); override;
```

```
    property ShowDebug:        boolean    read _ShowDebug        write _ShowDebug;
    property Sleep:            integer    read _Sleep            write _Sleep;
    property NearClipping:     single     read _NearClipping     write _NearClipping;
    property FarClipping:      single     read _FarClipping      write _FarClipping;
  end;

  TgameVar = record
    sleep:            integer;
    speed:            real;
    gravity:          integer;
    cammode:          integer;
    keys:             TKeys;
    cams:             integer;
    Fog:              TFog;

    isAdmin:          boolean;
    showDebug:        boolean;
  end;

  TgameVariabeln = class(Tobject)
    //Setup
    VideoConfig:      TVideoConfig;
    AudioConfig:      TAudioConfig;
    KeyConfig:        TKeyConfig;
    MouseConfig:      TMouseConfig;
    MiscConfig:       TMiscConfig;

    constructor Create(iniPath: string);
    destructor Destroy; override;
    procedure setdefault;
    procedure LoadFromIni;
    procedure SaveToIni;
  end;

implementation

uses
  Mainunit;

{#########################################################################
#########
------------------------TAllConfig-------------------------------
#########################################################################
#########}
constructor TAllConfig.Create(__iniFile: string);
begin
  inherited Create;
  _iniFile := __iniFile;
  Load;
end;
```

```
destructor TAllConfig.Destroy;
begin
  Save;
  inherited Destroy;
end;

procedure TAllConfig.Load;
var
  Ini: TIniFile;
begin
  ini := TIniFile.Create(iniFile);
  try
    LoadIndividual(Ini);
  finally
    ini.Free;
  end;
end;

procedure TAllConfig.Save;
var
  Ini: TIniFile;
begin
  ini := TIniFile.Create(iniFile);
  try
    SaveIndividual(Ini);
  finally
    ini.Free;
  end;
end;

{#########################################################################
#########
------------------------TVideoConfig-------------------------------
#########################################################################
#########}
procedure TVideoConfig._SetBitDepthIndex(Value: Integer);
begin
  _BitDepthIndex := Value;

end;

procedure TVideoConfig._SetResolutionIndex(Value: Integer);
begin
  _ResolutionIndex := Value;

end;

procedure TVideoConfig._SetShader(Value: boolean);
begin
 if _Shader <> Value then
 begin
   _Shader := Value;
```

```
   InitShaders;
 end;
end;

procedure TVideoConfig.LoadIndividual(Ini: TIniFile);
begin
  _VerticalSync    := ini.ReadBool    ('TVideoConfig', 'VerticalSyncronisation',  false);
  _Shader          := ini.ReadBool    ('TVideoConfig', 'EnableShader',            false);
  _WindowMode      := ini.ReadBool    ('TVideoConfig', 'GoWindowMode',            false);
  _Gamma           := ini.ReadFloat   ('TVideoConfig', 'Gamma',                   1);
  _ResolutionX     := ini.ReadInteger ('TVideoConfig', 'ResolutionX',             800);
  _ResolutionY     := ini.ReadInteger ('TVideoConfig', 'ResolutionY',             600);
  _GraphicQuality  := ini.ReadInteger ('TVideoConfig', 'GraphicQuality',          1);
  _BitDepth        := ini.ReadInteger ('TVideoConfig', 'BitDepth',                1);
end;

procedure TVideoConfig.SaveIndividual(Ini: TIniFile);
begin
  ini.WriteBool    ('TVideoConfig', 'VerticalSyncronisation',  _VerticalSync);
  ini.WriteBool    ('TVideoConfig', 'EnableShader',            _Shader);
  ini.WriteBool    ('TVideoConfig', 'GoWindowMode',            _WindowMode);
  ini.WriteFloat   ('TVideoConfig', 'Gamma',                   _Gamma);
  ini.WriteInteger ('TVideoConfig', 'ResolutionX',             _ResolutionX);
  ini.WriteInteger ('TVideoConfig', 'ResolutionY',             _ResolutionY);
  ini.WriteInteger ('TVideoConfig', 'GraphicQuality',          _GraphicQuality);
  ini.WriteInteger ('TVideoConfig', 'BitDepth',                _BitDepth);
end;

{##########################################################################
#########
-----------------------TAudioConfig------------------------------
##########################################################################
#########}
procedure TAudioConfig.LoadIndividual(Ini: TIniFile);
begin
  _Sound           := ini.ReadBool    ('TAudioConfig', 'Sound',                 true);
  _Music           := ini.ReadBool    ('TAudioConfig', 'Music',                 true);
  _SoundVolume     := ini.ReadFloat   ('TAudioConfig', 'SoundVolume',           0.75);
  _MusicVolume     := ini.ReadFloat   ('TAudioConfig', 'MusicVolume',           0.75);
end;

procedure TAudioConfig.SaveIndividual(Ini: TIniFile);
begin
  ini.WriteBool    ('TAudioConfig', 'Sound',                 _Sound);
  ini.WriteBool    ('TAudioConfig', 'Music',                 _Music);
  ini.WriteFloat   ('TAudioConfig', 'SoundVolume',           _SoundVolume);
  ini.WriteFloat   ('TAudioConfig', 'MusicVolume',           _MusicVolume);
end;

{##########################################################################
#########
------------------------TKeyConfig-------------------------------
```

```
##################################################################
#########}
procedure TKeyConfig.LoadIndividual(Ini: TIniFile);
begin
  //
end;

procedure TKeyConfig.SaveIndividual(Ini: TIniFile);
begin
  //
end;

{##################################################################
#########
------------------------TMouseConfig-----------------------------
##################################################################
#########}
procedure TMouseConfig.LoadIndividual(Ini: TIniFile);
begin
  _MouseSpeed      := ini.ReadFloat   ('TMouseConfig', 'MouseSpeed',            0.5);
  _MouseAcc        := ini.ReadFloat   ('TMouseConfig', 'MouseAcc',             0.5);
  _InvertMouse     := ini.ReadBool    ('TMouseConfig', 'InvertMouse',          false);
end;

procedure TMouseConfig.SaveIndividual(Ini: TIniFile);
begin
  ini.WriteFloat   ('TMouseConfig', 'MouseSpeed',           _MouseSpeed);
  ini.WriteFloat   ('TMouseConfig', 'MouseAcc',             _MouseAcc);
  ini.WriteBool    ('TMouseConfig', 'InvertMouse',          _InvertMouse);
end;

{##################################################################
#########
-------------------------TMiscConfig-----------------------------
##################################################################
#########}
procedure TMiscConfig.LoadIndividual(Ini: TIniFile);
begin
  _ShowDebug       := ini.ReadBool    ('TMiscConfig', 'ShowDebug',            false);
  _Sleep           := ini.ReadInteger ('TMiscConfig', 'Sleep',                5);
  _NearClipping    := ini.ReadFloat   ('TMiscConfig', 'NearClipping',         0.5);
  _FarClipping     := ini.ReadFloat   ('TMiscConfig', 'FarClipping',          300);
end;

procedure TMiscConfig.SaveIndividual(Ini: TIniFile);
begin
  ini.WriteBool    ('TMiscConfig', 'ShowDebug',            _ShowDebug);
  ini.WriteInteger ('TMiscConfig', 'Sleep',                _Sleep);
  ini.WriteFloat   ('TMiscConfig', 'NearClipping',         _NearClipping);
  ini.WriteFloat   ('TMiscConfig', 'FarClipping',          _FarClipping);
end;
```

```
{#########################################################################
#########
------------------------TGameVariabeln----------------------------
#########################################################################
#########}
constructor TGameVariabeln.Create(iniPath: string);
begin
  inherited Create;
  VideoConfig := TVideoConfig.Create(iniPath);
  AudioConfig := TAudioConfig.Create(iniPath);
  KeyConfig   := TKeyConfig.Create(iniPath);
  MouseConfig := TMouseConfig.Create(iniPath);
  MiscConfig  := TMiscConfig.Create(iniPath);
end;

destructor TGameVariabeln.Destroy;
begin
  VideoConfig.Destroy;
  AudioConfig.Destroy;
  KeyConfig.Destroy;
  MouseConfig.Destroy;
  MiscConfig.Destroy;
  inherited Destroy;
end;

procedure TGameVariabeln.setdefault;
var
  i, j:     integer;
begin
  game.gameVar.Fog.FogColor.R     :=    1;
  game.gameVar.Fog.FogColor.G     :=    1;
  game.gameVar.Fog.FogColor.B     :=    1;
  game.gameVar.Fog.FogColor.A     :=    0;
  game.gameVar.Fog.fogstart       :=   15;
  game.gameVar.Fog.fogend         :=   100;
  game.gameVar.Fog.fogdensity     :=    0.01;
  game.gameVar.Fog.fogmode        :=    1;     //GL_LINEAR = 1; GL_EXP = 2; GL_EXP2 = 3;
  game.gameVar.Fog.Enabled        :=    true;

  with game.gameVar do
  begin
    sleep          := 10;
    speed          := 0.5;
    gravity        := 100;
    cammode        := 0;
    cams           := 3;
    showdebug      := true;
    with keys do
    begin
      forw.which         :=  SDLK_w         ;
      backw.which        :=  SDLK_s         ;
      left.which         :=  SDLK_a         ;
```

```
    right.which        :=  SDLK_d        ;
    jump.which         :=  SDLK_Space    ;
    camchange.which    :=  SDLK_F1       ;
    exit.which         :=  SDLK_ESCAPE   ;
    pause.which        :=  SDLK_Pause    ;
    fullscreen.which   :=  SDLK_RETURN   ;

    forw.state         :=  false         ;
    backw.state        :=  false         ;
    left.state         :=  false         ;
    right.state        :=  false         ;
    jump.state         :=  false         ;
    camchange.state    :=  false         ;
    exit.state         :=  false         ;
    pause.state        :=  false         ;
    fullscreen.state   :=  false         ;
  end;
 end;
end;

procedure TgameVariabeln.LoadFromIni;
var
 Ini: TIniFile;
begin
 try
  Ini:=TIniFile.Create(ExtractFilePath(ParamStr(0))+'config.ini');

  game.gameVar.sleep          := ini.ReadInteger( 'game.gameVar',      'sleep',          10);
  game.gameVar.gravity        := ini.ReadInteger( 'game.gameVar',      'gravity',        100);
  game.gameVar.cammode        := ini.ReadInteger( 'game.gameVar',      'cammode',        0);
  game.gameVar.cams           := ini.ReadInteger( 'game.gameVar',      'cams',           3);

  game.gameVar.Speed          := ini.Readfloat(   'game.gameVar',      'Speed',          0.5);

  game.gameVar.Fog.FogColor.r := ini.Readfloat(   'game.gameVar.Fog',  'red',            1);
  game.gameVar.Fog.FogColor.b := ini.Readfloat(   'game.gameVar.Fog',  'blue',           1);
  game.gameVar.Fog.FogColor.g := ini.Readfloat(   'game.gameVar.Fog',  'green',          1);
  game.gameVar.Fog.FogColor.a := ini.Readfloat(   'game.gameVar.Fog',  'alpha',          1);
  game.gameVar.Fog.fogstart   := ini.Readfloat(   'game.gameVar.Fog',  'fogstart',       10);
  game.gameVar.Fog.fogend     := ini.Readfloat(   'game.gameVar.Fog',  'fogend',         100);
  game.gameVar.Fog.fogdensity := ini.Readfloat(   'game.gameVar.Fog',  'fogdensity',     0.01);
  game.gameVar.Fog.fogmode    := ini.Readinteger( 'game.gameVar.Fog',  'fogmode',        1);
  game.gameVar.Fog.enabled    := ini.Readbool(    'game.gameVar.Fog',  'fogenabled',     false);


 finally
  Ini.Free;
 end;
end;

procedure TgameVariabeln.SaveToIni;
var
```

```
  Ini: TIniFile;
begin
  try
    Ini:=TIniFile.Create(ExtractFilePath(ParamStr(0))+'config.ini');

    ini.WriteInteger('game.gameVar',      'sleep',          game.gameVar.sleep);
    ini.WriteInteger('game.gameVar',      'gravity',        game.gameVar.gravity);
    ini.WriteInteger('game.gameVar',      'cammode',          game.gameVar.cammode);
    ini.WriteInteger('game.gameVar',      'cams',           game.gameVar.cams);

    ini.Writefloat(  'game.gameVar',      'Speed',          game.gameVar.Speed);

    ini.Writefloat(  'game.gameVar.Fog',  'red',            game.gameVar.Fog.FogColor.r);
    ini.Writefloat(  'game.gameVar.Fog',  'blue',           game.gameVar.Fog.FogColor.b);
    ini.Writefloat(  'game.gameVar.Fog',  'green',          game.gameVar.Fog.FogColor.g);
    ini.Writefloat(  'game.gameVar.Fog',  'alpha',          game.gameVar.Fog.FogColor.a);
    ini.Writefloat(  'game.gameVar.Fog',  'fogstart',       game.gameVar.Fog.fogstart);
    ini.Writefloat(  'game.gameVar.Fog',  'fogend',         game.gameVar.Fog.fogend);
    ini.Writefloat(  'game.gameVar.Fog',  'fogdensity',     game.gameVar.Fog.fogdensity);
    ini.Writeinteger('game.gameVar.Fog',  'fogmode',          game.gameVar.Fog.fogmode);
    ini.Writebool(   'game.gameVar.Fog',  'fogenabled',     game.gameVar.Fog.enabled);

    //Keys
  finally
    Ini.Free;
  end;
end;

end.
```

8. GUI

```
unit GUI;
{ Eine GUI gemacht von David Halter (Gaukler). Mein erstes Objektorientiertes Projekt,
  ist also noch nicht perfekt objektorientiert, aber das ganze ist schon ziemlich leicht in andere
Projekte zu importieren...
  Die GUI ist überhaupt nicht speed - orientiert.
}

//Release 1.0
{Developer Blog:
*  1.11.06:   First Relase, 7 Grundverschiedene Elemente und 8 verschiedene Mouse/Key
Routinen.
*  4.11.06:   Maximize verbessert, da gab es zwei bugs, man konnte das fenster maximiert
verschieben und
           Maximiert wurde auch, wenn nur ein downclick erfolgte.
*  9.11.06:   SetPointers so verändert, dass jetzt die Pointer immer richtig gesetzt sind.
*  11.11.06:  deinitialisierung verbessert, wegen memory leaks
*  13.11.06:  Objektorientierte Struktur verbessert, onDestroyEvent geadded.
           Mouseevents grossteils umprogrammiert, zwecks makros auf rechter maustaste.
*  15.11.06:  PopUpMenu, Textlist und Frames soweit vorbeireitet, dass nur noch die Events
gemacht werden müssen.
```

* 20.11.06: Images funktionieren nun vollständig, mit Blending und Farben.
Neue Art des Schriften ladens hinzugefügt, die zwar schon funktioniert, aber nicht sonderlich schön aussieht.
* 23.11.06: Mouse Events wieder fest umgeschrieben
PopUpMenu funktioniert in Grundzügen
* 25.11.06: Edit funktioniert nun vollständig, man kann nun schreiben usw
neue Art Schriften zu laden hinzugefügt, um die Breiten der Schriften Edit tauglich zu machen.
PopUpMenu zum grössten teil funktionierend gemacht
Makros geschrieben für einige funktionen, zb um mit einem mausklick die Progressbar-länge zu verändern.
* 27.11.06: Textlist läuft nun.
Texturen von PopUpMenu/Combobox verbessert.
* 28.11.06: Skins überarbeitet, nun kann man jpgs tgas und bmps laden.
* 30.11.06: EditField soweit fertiggestellt; Drag-Bug behoben
* 31.11.06: EditField besitzt nun einige Funktionen mehr, man kann es fast wie ein Word Dokument bedienen.
Maximize - Bug behoben, einmal wurde der Boolean Wert verkehrt herum abgerufen.
CaptionBar verschönert.
* 01.12.06: Nach nun genau einem Monat, nach dem First Release, ist das Release 1.0 draussen.
Es sind etwa 1400 Zeilen dazu gekommen und jede Menge neue Elemente.
Das was sicher noch kommt wird eine Ladefunktion sein, allerdings wird diese bis auf weiteres warten müssen,
zuerst muss die GUI in Terrorist's Revenge ausgiebig getestet werden.
* 09.12.06: Nun habe ich mich ein bisschen in GIMP eingearbeitet (welches übrigens ein sehr gutes Zeichenprogramm ist) um
neue GUI - Texturen machen zu können. Ausserdem habe ich das Rendern der Buttons abgeändert.
* 14.12.06: Z-Abfragen funktionieren nun viel besser, ausserdem habe ich angefangen eine Speichermethode zu implementieren...
* 25.12.06: Streaming fertiggestellt (save & load) aber Bugs müssen noch ausgemerzt werden.
* 26.12.06: Beinahe alle Pointer abgeschafft, weil diese das ganze nur kompliziert machten und Bug-anfällig.
* 29.12.06: Streaming läuft nun perfekt, alle Bugs behoben.
Comboboxes und PopUpMenus hatten einen Bug, wo die Zählvariabeln i und j vertauscht wurden (in onMousedown und onMouseup). Fixed.
Durch Streaming wurde nun Version 1.1 erreicht. Das nächste Ziel wird der Editor sein.
PopUpMenus werden nun beim klicken auf irgendetwas geschlossen.
Das aktive Fenster wird nun im Z Wert höher gestellt wie normal.
* 31.12.06: Z wurde vielfach falsch gesetzt. Fixed.
* 01.01.07: Space ist nun der bezeichner für den Abstand zwischen den Combobox-Zeilen, nicht mehr Height.
* 02.01.07: Der Editor steht nun, man muss nur noch ein paar sachen verbessern.
Edits und EditFields gaben eine Exception von sich, wenn man an den falschen Ort klickte.
Update auf Version 1.2.
* -25.02.07: Viele kleine Bugs behoben.
* -01.03.07: Nun werden die Fenster immer schön im Vordergrund gehalten, wenn sie im Vordergrund sind (verbesserte Z-Order).
Nach nochmal 100 Bugs ist die GUI eigentlich fertig.
Ich habe es nun auch endlich geschafft ALLE Pointer loszuwerden und somit auch die eklige Prozedur Setpointers komplett weg ist.

Damit sind auch wieder 100-200 Zeilen gelöscht worden.
* -01.07.07: SDL-fähig gemacht!
Anzeige des blinkenden Cursors bei Texten ändert sich nicht mehr durch diese Unit, wird durch die Engine erledigt. (Terr. Revenge)
}

{TODO:
* DONE_Bug: Frames bewegen geht nicht wirklich.
* DONE_Editor:
DONE_- Man sollte irgendwie die Anzahl der Frames bearbeiten können
DONE_- Man sollte auch die PopUpMenuItems und ComboboxItems verändern können. Der Rest stimmt.
* GUI testen! //bis jetzt funktioniert alles, was funktionieren soll.
* evtl. scrollbar //wird wohl noch warten müssen, wenns überhaupt kommt.

* DONE_PopupMenu //funktioniert
* DONE_Images //dito
* DONE_EditField //grundfunktionen laufen.
* DONE_TextField //läuft
* DONE_combobox //so weit so gut
* DONE_frames //läuft

* DONE_RadioButton überarbeiten, unter anderem onMouseUp, die gruppen irgendwie einbeziehn. //DONE, sollte theoretisch laufen, aber noch testen
* DONE_Rendern/MouseEvents des Edits //theoretisch erledigt, aber sicher noch falsche positionen.
* DONE_Button Bug beheben, mit active und pressed und den mouseevents //getestet und funktioniert
* DONE_width/height anpassung, der label... //auch gemacht, aber auch falsche positionen.
* DONE_minimize, maximize usw... für die fenster //funkioniert perfekt.
* DONE_Schrift im allgemeinen verbessern... // so weit so gut
* DONE_Texturen kehren!!!! //vollständig gemacht, aber workaround.
* DONE_Bug: Beim draggen von Elementen (nicht fenster) gibts eine exception. //lag daran, das lastmousemove benutzt wurde und nicht lastmousedown...
* DONE_Bug: Maximize geht nicht mehr. In der Abfrage ChangeWindowState wurde der Boolean Wert verkehrt abgefragt, jetzt habe ich das ganze auch noch geändert, damits intuitiver richtig gemacht wird...
* DONE_Save/Load Windows wenn der rest läuft.
}

{Hints:
* Die Events des PopUpMenus funktionieren absichtlich nicht, es funktionieren nur die Events der einzelnen kleinen Fenster...
* Die RadioButtonGroups besitzen kein Element 0, darauf zuzugreifen bringt nichts.
* Fenster die geladen wurden haben zwingend keine Events, das heisst man muss sie neu dazu tun.
* Fenster können nicht kopiert werden, sie können aber abgespeichert und geladen werden. Dann haben sie allerdings keine Events.

* Das Rendern mit der GUI ist ganz leicht und funktioniert so:

```
  >< glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  >< GUIclass.GoOrtho(Panel1.Width,Panel1.Height, -10, 10);
  >< GUIclass.Render;
  >< GUIclass.ExitOrtho(60,ClientWidth/ClientHeight,1,128);
  >< SwapBuffers(DC);
```

Dies kann man einfach in eine Schleife setzen und immer machen lassen. Die Events müssen allerdings separat abgefangen werden.

```
}

{
objecttypes:
  *0:    Buttons
  *1:    Text
  *2:    Panels
  *3:    ProgressBars
  *4:    Edits
  *5:    Checkboxes
  *6:    RadioButtons
  *7:    Windows
  *8:    minimizeButton
  *9:    maximizeButton
  *10:   closeButton

  new:
  *11:   EditField
  *12:   TextField
  *13:   PopUpMenu
  *14:   Combobox
  *15:   Frames
  *16:   Image
  *17:   ScrollBar      ???

  *30:   PopUpMenuItem
  *31:   ComboboxItem

  *50:   Text (ohne Window)
}

{Tiefen der Objekte:
* Button: 1.1
* Text: 1.2
* checkbox: 0.9
* progressbar: 0.2
* panel: 0.1
* edit 1
* radiobutton: 0.8

* Editlist: 0.15
* Textlist: 0.14
* PopUpMenu: 1.25
* Combobox: 0.7
```

```
* Frames:   0.121
* Image:   0.13
}

interface

//ob errors angezeigt werden oder nicht:
{$define showerrors}

//ob mit SDL oder ohne gearbeitet wird:
{$define sdl}

uses
  dglOpenGL, textures, Classes, Dialogs, sysutils,
  FileTypes;  //Eigene Unit...

const
  VERSION = '1.2';

type
  TWindow = class;
  TPopupMenu = class;
  TGUIObject = class;

  TColor = record
    r,g,b, a: single;
  end;

  TPoint = record
    x,y: integer;
  end;

  TFont = record
    path:            string;
    graphic:          cardinal;
    list:            integer;
    name:             string;
    width:           integer;
    height:          integer;
    used:            boolean;
  end;

  TSkin = record
    path:            string;

    panel:            cardinal;
    edit:            cardinal;
    captionbar:       cardinal;
    progress_ground:  cardinal;
    progressbar:      cardinal;
    ground:          cardinal;
    editfield:       cardinal;
```

```
  button:          cardinal;
  button_c:        cardinal;
  button_m:        cardinal;

  frame:           cardinal;
  frame_c:         cardinal;

  check:           cardinal;
  check_m:         cardinal;
  check_c:         cardinal;
  check_m_c:       cardinal;

  radio:           cardinal;
  radio_m:         cardinal;
  radio_c:         cardinal;
  radio_m_c:       cardinal;

  minimize:        cardinal;
  minimize_m:      cardinal;
  normalize:       cardinal;
  normalize_m:     cardinal;
  maximize:        cardinal;
  maximize_m:      cardinal;
  close:           cardinal;
  close_m:         cardinal;
  default:         cardinal;

  combobox:        cardinal;
  combobox_button: cardinal;
  combo_ground:    cardinal;

  used:            boolean;
end;

Tonclick =        procedure(X,Y: integer; Element: TGUIObject; Button: integer);
TonMousedown =    procedure(X,Y: integer; Element: TGUIObject; Button: integer);
TonMouseup =      procedure(X,Y: integer; Element: TGUIObject; Button: integer);
Tonmouseover =    procedure(X,Y: integer; Element: TGUIObject);
Tafterdrag =      procedure(X,Y: integer; Element: TGUIObject; Button: integer);
TonDrag =         procedure(X,Y: integer; Element: TGUIObject; Button: integer);

Tonkeydown =      procedure(Key: Word; Element: TGUIObject);
Tonkeyup =        procedure(Key: Word; Element: TGUIObject);

TonDestroy =      procedure(Element: TGUIObject);

TGUIObject = class(TObject)
private
  _X:             integer;
  _Y:             integer;
  _Z:             single;        //Tiefe des Objektes, wichtig was im Vordergrund liegt.
```

```
  _Width:          integer;
  _Height:         integer;
  _Visible:        boolean;       //Sichtbarkeitsüberprüfung (auch Code - Optimierung)

  haspopupMenu:    boolean;       //PopUpMenus...
  FPopUpMenu:      TPopUpMenu;

  //die Events - ob es einen Event überhaupt gibt (Wichtig für Geschwindigkeitsoptimierung
  onclickevent:    boolean;
  onMousedownevent: boolean;
  onMouseupevent:  boolean;
  onkeydownevent:  boolean;
  onkeyupevent:    boolean;
  onDragEvent:     boolean;
  onDestroyEvent:  boolean;
  afterDragEvent:  boolean;  //ob nach den drags etwas ausgeführt wird...

  //die einzelnen Events...
  Fonmouseoverevent: boolean;
  Fonclick:        Tonclick;
  FonMousedown:     TonMousedown;
  FonMouseup:      TonMouseup;
  Fonmouseover:     Tonmouseover;
  Fonkeydown:      Tonkeydown;
  Fonkeyup:        Tonkeyup;
  Fafterdrag:      Tafterdrag;
  FonDestroy:      TonDestroy;
  FonDrag:         TonDrag;

  procedure changeonmouseover(Value: boolean);
  procedure setonclick(Value: Tonclick);
  procedure setMousedown(Value: TonMousedown);
  procedure setMouseup(Value: TonMouseUp);
  procedure setMouseover(Value: TonMouseOver);
  procedure setkeydown(Value: TonKeydown);
  procedure setkeyup(Value: TonKeyUp);
  procedure setafterdrag(Value: Tafterdrag);
  procedure setondrag(Value: TonDrag);
  procedure setonDestroy(Value: TonDestroy);
  procedure setPopUpMenu(Value: TPopUpMenu);

  procedure setX(Value: integer);
  procedure setY(Value: integer);
  procedure setZ(Value: single);
  procedure SetWidth(Value: integer);
  procedure SetHeight(Value: integer);
  procedure SetVisible(Value: boolean);

  procedure RenderExact(X,Y,Width,Height: integer; Z, divi: single; plus: integer);

  procedure LoadFromStream_Individual(S: TBinaryFile); virtual;
  procedure SaveToStream_Individual(S: TBinaryFile); virtual;
```

```
  protected
    //ob Mouse- bzw Keyevents ausgeführt werden
  public
    skin:           integer;        //Die Skin - Art, zb Windoof.
    font:           integer;        //Die Schriftart
    Parent:          TWindow;        //Das Fenster, dem das Objekt untergeordnet ist. Wenn
objekttyp = fenster dann zeigt es auf sich selbst.
    Name:            string;         //nicht wichtig, aber da um Verwechslungen zu vermeiden
    dragevent:       boolean;        //ob ein Drag am laufen ist
    active:          boolean;        //für die key events - wann ein objekt zuletzt gedrückt wurde...
    tag:            integer;        //für alles und gar nichts, einfach ein zusätzlicher parameter der
gebraucht werden kann wie man will.
    objecttype:      integer;        //was für ein typ das objekt ist...
    Index:           integer;        //der index im array drin...
    used:            boolean;        //ob ein Objekt benützt ist

    //ob die Mouse auf einem Element liegt
    function  MouseonElement(mX,mY: integer): boolean; virtual;

    //Lade & Speicher Prozeduren
    procedure SaveToStream(S: TBinaryFile); virtual;
    procedure LoadFromStream(S: TBinaryFile); virtual;

    //ob es onmouseoverevents gibt. Da es vielfach keine gibt und die maus immer viel bewegt
wird, muss so wenig abgefragt werden...
    property onmouseoverevent: boolean read Fonmouseoverevent write changeonmouseover
default false;
    //Die Proceduren die ausgeführt werden, wenn zb ein Klick auf ein Objekt stattfindet
    property onMouseDown:  TonMousedown read FonMousedown write setMousedown;
    property onMouseUp:    TonMouseup read FonMouseUp write setMouseup;
    property onMouseOver:  Tonmouseover read FonMouseover write setMouseover;
    property onKeyDown:    Tonkeydown read FonKeydown write setkeydown;
    property onKeyUp:      Tonkeyup read FonKeyUp write setkeyup;
    property onDrag:       Tondrag read Fondrag write setondrag;
    property afterDrag:    Tafterdrag read Fafterdrag write setafterdrag;
    property onClick:      Tonclick read Fonclick write setonclick;
    property onDestroy:    TonDestroy read FonDestroy write setonDestroy;

    property X:           integer read _X write setX;
    property Y:           integer read _Y write setY;
    property Z:           single read _Z write setZ;
    property Width:        integer read _Width write SetWidth;
    property Height:       integer read _Height write SetHeight;
    property Visible:      boolean read _Visible write SetVisible;
    property PopUpMenu:    TPopUpMenu read FPopUpMenu write setPopUpMenu;
  end;

  TDoSomething  =     procedure(Element: TGUIObject);

  TGUIField = class(TGUIObject)
  private
    function  GetText: string;
```

```
    procedure Update(Value :string);
  public
    Stringlist: array of string;
    procedure MakeNewLine(which: integer);
    procedure DeleteLine(which: integer);
    property Text: string read GetText write Update;     //sollte später nicht mehr read von FText
sein...
  end;

  TImage = class(TGUIObject)
    blending: boolean;
    sfactor:  cardinal;   //Blending faktoren
    dfactor:  cardinal;
    graphic:  cardinal;
    graphicpath: string;
    Color:    TColor;
    procedure Render;
    procedure SaveToStream_Individual(S: TBinaryFile);  override;
    procedure LoadFromStream_Individual(S: TBinaryFile); override;
  end;

  TText = class(TGUIObject)
  private
    FText: string;
  public
    color:      TColor;
    procedure Update(Value :string);
    property Text: string read FText write Update;
    procedure SaveToStream_Individual(S: TBinaryFile);  override;
    procedure LoadFromStream_Individual(S: TBinaryFile); override;
  end;

  TTextField = class(TGUIField)
    Space:          integer;      //der abstand der zeilen...
    TextColor:      TColor;
    procedure Render;
    procedure SaveToStream_Individual(S: TBinaryFile);  override;
    procedure LoadFromStream_Individual(S: TBinaryFile); override;
  end;

  TButton = class(TGUIObject)
    plusx:      integer;
    plusy:      integer;
    Caption:    string;
    Pressed:    boolean;
    TextColor:  TColor;
    procedure Render;
    procedure SaveToStream_Individual(S: TBinaryFile);  override;
    procedure LoadFromStream_Individual(S: TBinaryFile); override;
  end;

  TPopupMenuItem = class(TGUIObject)
```

```
  Caption:  string;
  ParentMenu: integer;
  mParent: TPopUpMenu;
 end;

 TPopupMenu = class(TGUIObject)
  PopupMenuItems: array of TPopupMenuItem;
  opened:       boolean;
  whichchecked: integer;
  TextColor:    TColor;
  plusx:      integer;
  plusy:      integer;
  procedure Render;
  function MouseOnElement(mX,mY: integer): boolean; override;
  function AddPopUpItem(Caption: string): integer;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TMinimizeButton = class(TGUIObject)
  pressed:    boolean;
  minimized:  boolean;
 end;

 TMaximizeButton = class(TGUIObject)
  pressed:    boolean;
  maximized:  boolean;
 end;

 TCloseButton = class(TGUIObject)
  pressed:    boolean;
 end;

 TPanel = class(TGUIObject)
  procedure Render;
 end;

 TFrame = record
  Window:     TWindow;
 end;

 TFrames = class(TGUIObject)
  Frames:          array of TFrame;
  CaptionBarHeight: integer;
  whichchecked:     integer;
  tabwidth:        integer;
  plusx:           integer;
  plusy:           integer;
  TextColor:        TColor;
  procedure Render;
  function  AddFrame(Window: TWindow): integer;
  procedure MoveFrame(dX,dY: integer);
```

```
  function MouseOnElement(mX,mY: integer): boolean; override;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TProgressBar = class(TGUIObject)
  progress: single;    //[0..1]
  procedure Render;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TEdit = class(TGUIObject)
  Text:          string;
  TextColor:      TColor;
  MousePosition:  integer;
  plusx:      integer;
  plusy:      integer;
  procedure Render;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TEditField = class(TGUIField)     //unvollständig - überarbeiten
  readonly:       boolean;        //ob man nur lesen, oder eben auch schreiben darf...
  space:         integer;       //abstand zwischen den zeilen...
  line:          integer;       //für cursor
  position:       integer;       //...
  Textcolor:      TColor;
  plusx:      integer;
  plusy:      integer;
  procedure Render;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TCheckbox = class(TGUIObject)
  Checked:  Boolean;
  Pressed:  boolean;
  procedure Render;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
 end;

 TComboBoxItem = class(TGUIObject)          //muss noch gemacht werden
  Caption:  string;
 end;

 TCombobox = class(TGUIObject)
  ComboboxItems:  array of TComboBoxItem;
  Opened:         boolean;
  Whichchecked:   integer;
```

```
  mouseon:        integer;
  Caption:        string;
  TextColor:      TColor;
  plusx:          integer;
  plusy:          integer;
  space:          integer;
  procedure Render;
  function MouseOnElement(mX,mY: integer): boolean; override;
  function  AddComboBoxItem(Caption: string): integer;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
end;

TRadioButtonGroup = class;

TRadioButton = class(TGUIObject)
  Checked:      Boolean;
  pressed:      boolean;
  ParentGroup:  TRadioButtonGroup;
  Group:        integer;
  GroupIndex:   integer;
  procedure Render;
  procedure SaveToStream_Individual(S: TBinaryFile);  override;
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
end;

TRadioButtonGroup = class(TObject)
  whichchecked: integer;
  RadioButtons: array of TRadioButton;
end;

TChildWindow = record
  Child: TWindow;
  Index: Integer;
  used: boolean;
end;

TWindow = class(TGUIObject)
private
  //Die Hintergrundgrafik
  graphicpath:      string;
  //für das Maximieren von Fenstern.
  oldPoint:         TPoint;
  oldWidthHeight:   TPoint;
  //Pointer auf sich selbst um dem self. auszuweichen.
  Pself:            TWindow;
  //ob ein element onmouseover hat:
  hasonmouseover:   boolean;
  //ob das Fenster Childs hat...
  hasChilds:        boolean;
  //ob es ein Tab ist.
  isTab:            boolean;
```

```
  //die überarbeitete MouseonElement für Fenster.
  function  MouseonElement(mX,mY: integer): boolean; override;
  //Das Rendern des Windows
  procedure Render;
  //Die überladenen Methoden um aus Streams zu laden & speichern.
  procedure LoadFromStream_Individual(S: TBinaryFile); override;
  procedure SaveToStream_Individual(S: TBinaryFile); override;
public
  //Aussehen
  Color:           TColor;
  FontColor:       TColor;
  background:      boolean;   //ob es einen hinterrgrund gibt.
  graphic:         cardinal;

  //Die CaptionBar
  CaptionBar:      boolean;
  CaptionBarHeight: integer;
  Caption:         string;

  MinimizeButton:   TMinimizeButton;
  MaximizeButton:   TMaximizeButton;
  CloseButton:      TCloseButton;

  //Ob es ein Parent gibt, wichtig damit nicht 2-n mal gerendert wird.
  Child:           boolean;

  ztemp:           double;   //wichtig für die Tiefen Einstellung.

  //Die Komponenten
  Buttons:          Array of TButton;
  Text:            Array of TText;
  Panels:           Array of TPanel;
  ProgressBars:      Array of TProgressBar;
  Edits:           Array of Tedit;
  Checkboxes:        Array of TCheckBox;
  RadioButtons:      Array of TRadioButton;
  RadioButtonGroups:  Array of TRadioButtonGroup;
  ChildWindows:      Array of TChildWindow;
  Images:           Array of TImage;            //new
  PopupMenus:        Array of TPopupMenu;        //new
  Comboboxes:        Array of TCombobox;         //new
  Frames:           Array of TFrames;           //new
  TextFields:        Array of TTextfield;        //new
  EditFields:        Array of TEditfield;        //new

  //Das Hinzufügen von Neuen Komponenten
  function  AddButton(btnX, btnY, btnWidth, btnHeight : Integer; btnCaption : String): integer;
  function  AddText(tX, tY : Integer; tText : String; tColor: TColor): integer;
  function  AddPanel(pX, pY, pWidth, pHeight : Integer): integer;
  function  AddProgressBar(pX, pY, pWidth, pHeight : Integer): integer;
  function  AddEdit(pX, pY, pWidth, pHeight : Integer; pText: String): integer;
```

```
    function  AddCheckbox(cbX, cbY, cbwidth, cbheight : Integer; cbChecked : Boolean): integer;
    function  AddRadioButton(rbX, rbY, rbwidth, rbheight, rbGroup : Integer; rbChecked :
Boolean): integer;
    function  AddRadioButtonToGroup(GroupIndex: integer; RadioButton: TRadioButton): integer;
    function  AddRadioButtonGroup: integer;
    function  AddChildWindow(pChild : TWindow): integer;
    function  AddImage(iX,iY,iWidth,iHeight: integer; iBlending: boolean; iSfactor, iDfactor:
Cardinal; igraphic: string): integer;
    function  AddPopupMenu(pWidth, pHeight: integer; firstCaption: string): integer;
    function  AddCombobox(cX,cY,cWidth,cHeight: integer; firstCaption: string): integer;
    function  AddFrames(cX,cY,cWidth,cHeight: integer; firstWindow: TWindow): integer;
    function  AddTextField(tX,tY,tWidth,tHeight: integer; ttext: string; tfont: integer = -1): integer;
    function  AddEditField(tX,tY,tWidth,tHeight: integer; ttext: string; tfont: integer = -1): integer;

    procedure DoForEach(DoSomething: TDoSomething);
    procedure ChangeCaptionBarButtons;
    procedure FreeComponents;              //um Memory freizugeben...
    //Die überladenen Methoden um aus Streams zu laden & speichern.
    procedure LoadFromStream(S: TBinaryFile); override;
    procedure SaveToStream(S: TBinaryFile); override;
  end;

  TGUIclass = class(TObject)
  private
    //***Mouse***
    dragevent:       boolean;
    clickevent:      boolean;
    Mouseold:        TPoint;
    lastMouseDown:   TGUIObject;
    lastMouseMove:   TGUIObject;

    //Für die Reihenfolge der Windows.
    WinOrder:        array of integer;

    //keys
    keyisdown:       boolean;

    //Text Ausgabe
    function  GetKey(Value: Word): String;
    procedure PrintTextarray;
    procedure SetZforWindows;
    procedure Init;
    destructor Destroy;
  public
    //Für die Berechnung von sich ändernden Komponenten:
    Frame:           integer;

    //Key/mouse zeug
    lastkeydown:     word;
    lastMouseUp:     TGUIObject;
    CapsLock:        boolean;
    Shift:           boolean;
```

```
    Alt:            boolean;
    active:         TGUIObject;
    showcursor:     boolean;

    //Die Elemente der GUI Klasse
    Windows:        array of TWindow;
    Text:           array of TText;
    Skins:          array of TSkin;
    Fonts:          array of TFont;

    //Mouse Events
    procedure MouseMove(newX,newY: integer);
    procedure MouseDown(mX,mY, mButton: integer);
    procedure MouseUp(mX,mY, mButton: integer);
    //Key Events
    procedure KeyDown(Key: word);
    procedure KeyUp(Key: word);
    // das Printen von Fonts...
    procedure PrintFont(x, y: Integer; z: single; font: Integer; text : pchar);
    //Load Prozeduren
    function  LoadFont(index: integer; fontname: string): boolean;
    procedure LoadFont2(Index: integer; path: string; scale: single);
    //Add Funktionen
    function  AddWindow(wx,wy, wwidth,wheight, wSkin: integer; wcaption : string; wgraphic :
string = 'data\skins\default.tga'): integer;
    function  AddText(tx,ty,tz,tFont : integer; ttext: string; tColor: TColor) : integer;
    function  LoadSkinTexture(path: string; var Texture: cardinal): integer;
    function  AddSkin(pathdir: string): integer;
    function  AddFont(path: string; scale: single = 1): integer;
    //Delete Proceduren
    procedure DeleteWindow(index: integer);
    procedure DeleteText(index: integer);
    procedure DeleteSkin(index: integer);
    procedure DeleteFont(index: integer);
    //Laden & Speichern
    function LoadWindow(path: string; var Window: array of integer): boolean;
    function SaveWindow(path: string; const Window: array of integer): boolean;
    //Das Rendern der gesamten GUI
    procedure Render;

    procedure GoOrtho(Width, Height, zNear, zFar: double); //in den Ortho Mode wechseln...
    procedure ExitOrtho(FOV, aspect, znear, zfar: double); //aus dem Ortho Mode gehen...
  end;

procedure MousePostoProgressbar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure SetRedwithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure SetGreenwithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure SetBluewithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure ChangeWindowState(X,Y: integer; Element: TGUIObject; Button: integer = 1);
overload;
procedure Minimize(X,Y: integer; Element: TGUIObject; Button: integer = 1); overload;
procedure CloseWindow(Key: Byte; Element: TGUIObject); overload;
```

```
procedure CloseWindow(X,Y: integer; Element: TGUIObject; Button: integer = 1); overload;
procedure DestroyWindow(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure StopExe(Key: Byte; Element: TGUIObject); overload;
procedure StopExe(X,Y: integer; Element: TGUIObject; Button: integer = 1); overload;
procedure IncreaseAlpha(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure DecreaseAlpha(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure ShowPopUpMenu(X,Y: integer; Element: TGUIObject; Button: integer = 1);
procedure SetZforComponent(Element: TGUIObject);

procedure glRenderQuad(x1, y1, x2, y2: integer;  z: single); overload;
procedure glRenderQuad(x1, y1, x2, y2: integer; z, tx1, ty1, tx2, ty2: single); overload;


const
  //Colors with made with TColor
  cWhite:       TColor = ( r:  1; g:  1; b:  1; a:  1 );
  cBlack:       TColor = ( r:  0; g:  0; b:  0; a:  1 );
  cRed:         TColor = ( r:  1; g:  0; b:  0; a:  1 );
  cGreen:       TColor = ( r:  0; g:  1; b:  0; a:  1 );
  cBlue:        TColor = ( r:  0; g:  0; b:  1; a:  1 );
  cYellow:      TColor = ( r:  1; g:  1; b:  0; a:  1 );
  cMagenta:     TColor = ( r:  1; g:  0; b:  1; a:  1 );
  cCyan:        TColor = ( r:  0; g:  1; b:  1; a:  1 );
  cOrange:      TColor = ( r:  1; g:0.5; b:  0; a:  1 );
  cBrightBrown: TColor = ( r:  1; g:  1; b:0.7; a:  1 );
  cBrown:       TColor = ( r:0.5; g:0.25;b:  0; a:  1 );
  cGrey:        TColor = ( r:0.5; g:0.5; b:0.5; a:  1 );
  cDarkBlue:    TColor = ( r:  0; g:  0; b:0.5; a:  1 );
  cDarkRed:     TColor = ( r:0.5; g:  0; b:  0; a:  1 );
  cDarkGreen:   TColor = ( r:  0; g:0.5; b:  0; a:  1 );

var
  GUIclass: TGUIclass;

  Screen_width:  integer = 800;
  Screen_height: integer = 600;

implementation

var
  _counter:      integer;         //für die deinitialisierung

{##############################################################################
##################################
##############################################################################
##################################
onEventdo
##############################################################################
##################################
##############################################################################
##################################}
```

```
procedure MousePostoProgressbar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if X - (Element.parent.X + Element.X - 4) > 0 then
    element.Parent.ProgressBars[element.Index].progress := (X - (Element.parent.X + Element.X -
4))/Element.Width
  else element.Parent.ProgressBars[element.Index].progress := 0;
end;

procedure SetRedwithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if X - (Element.parent.X + Element.X - 4) > 0 then
    element.Parent.ProgressBars[element.Index].progress := (X - (Element.parent.X + Element.X -
4))/Element.Width
  else element.Parent.ProgressBars[element.Index].progress := 0;
  Element.parent.Color.r := element.Parent.ProgressBars[element.Index].progress;
end;

procedure SetGreenwithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if X - (Element.parent.X + Element.X - 4) > 0 then
    element.Parent.ProgressBars[element.Index].progress := (X - (Element.parent.X + Element.X -
4))/Element.Width
  else element.Parent.ProgressBars[element.Index].progress := 0;
  Element.parent.Color.g := element.Parent.ProgressBars[element.Index].progress;
end;

procedure SetBluewithProgressBar(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if X - (Element.parent.X + Element.X - 4) > 0 then
    element.Parent.ProgressBars[element.Index].progress := (X - (Element.parent.X + Element.X -
4))/Element.Width
  else element.Parent.ProgressBars[element.Index].progress := 0;
  Element.parent.Color.b := element.Parent.ProgressBars[element.Index].progress;
end;

procedure ChangeWindowState(X,Y: integer; Element: TGUIObject; Button: integer = 1);
overload;
var
  i: integer;
begin
  if button = 1 then
  begin
    if Element.Parent.MaximizeButton.maximized then
    begin
      //for i := 0 to High(Element.Parent.Frames) do
        //Element.Parent.Frames[i].MoveFrame(-Element.Parent.X,-Element.Parent.Y);
      Element.Parent.oldPoint.X := Element.Parent.X;
      Element.Parent.oldPoint.Y := Element.Parent.Y;
      Element.Parent.oldWidthHeight.X := Element.Parent.Width;
      Element.Parent.oldWidthHeight.Y := Element.Parent.Height;

      Element.Parent.X := 0;
```

```
      Element.Parent.Y := 0;
      Element.Parent.Width := Screen_Width;
      Element.Parent.Height := Screen_Height;
    end else
    begin
      Element.Parent.X := Element.Parent.oldPoint.X;
      Element.Parent.Y := Element.Parent.oldPoint.Y;
      Element.Parent.Width := Element.Parent.oldWidthHeight.X;
      Element.Parent.Height := Element.Parent.oldWidthHeight.Y;
      //for i := 0 to High(Element.Parent.Frames) do
        //Element.Parent.Frames[i].MoveFrame(Element.Parent.X,Element.Parent.Y);
    end;

    Element.Parent.ChangeCaptionBarButtons;
  end;
end;

procedure Minimize(X,Y: integer; Element: TGUIObject; Button: integer = 1); overload;
var
  i: integer;
  j: integer;
begin
  if button = 1 then
  begin
    Element.Parent.MinimizeButton.minimized := not Element.Parent.MinimizeButton.minimized;
    for i := 0 to High(element.parent.PopupMenus) do
      element.parent.PopupMenus[i].opened := false;
    for i := 0 to High(element.parent.Comboboxes) do
      element.parent.Comboboxes[i].opened := false;
    if Element.Parent.MinimizeButton.minimized then
    begin
      for i := 0 to High(element.Parent.Frames) do
        if element.Parent.Frames[i].used then
          for j := 0 to High(element.Parent.Frames[i].Frames) do
            element.Parent.Frames[i].Frames[j].Window.Visible := false;
    end else
    begin
      for i := 0 to High(element.Parent.Frames) do
        if element.Parent.Frames[i].used then
          //for j := 0 to High(element.Parent.Frames[i].Frames) do
          element.Parent.Frames[i].Frames[Element.Parent.Frames[i].whichchecked].Window.Visible
:= true;
    end;
  end;
end;

procedure CloseWindow(X,Y: integer; Element: TGUIObject; Button: integer = 1); overload;
begin
  if button = 1 then
  begin
    if Element.onDestroyEvent then Element.onDestroy(Element);
    Element.Parent.Visible := false;
```

```
  end;
end;

procedure CloseWindow(Key: Byte; Element: TGUIObject); overload;
begin
  if Element.onDestroyEvent then Element.onDestroy(Element);
  Element.Parent.Visible := false;
end;

procedure DestroyWindow(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if Button = 1 then
  begin
    if Element.onDestroyEvent then Element.onDestroy(Element);
    if Element.objecttype = 7 then
      Guiclass.DeleteWindow(Element.Index)
    else
      Guiclass.DeleteWindow(Element.Parent.Index);
  end;
end;

procedure StopExe(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if button = 1 then
    halt;
end;

procedure StopExe(Key: Byte; Element: TGUIObject);
begin
  halt;
end;

procedure IncreaseAlpha(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if button = 1 then
    if element.objecttype <> 7 then
      if element.Parent.Color.A <= 0.9 then element.Parent.Color.A := element.Parent.Color.A +
0.1
      else element.Parent.Color.A := 1;
end;

procedure DecreaseAlpha(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
  if button = 1 then
  begin
    if element.Parent.Color.A >= 0.1 then element.Parent.Color.A := element.Parent.Color.A - 0.1
    else element.Parent.Color.A := 0;
  end;
end;

procedure ShowPopUpMenu(X,Y: integer; Element: TGUIObject; Button: integer = 1);
begin
```

```
  if (Button = 3) and Element.hasPopUpMenu then
  begin
    Element.PopUpMenu.X := X + 2;
    Element.PopUpMenu.Y := Y + 2;
    Element.PopUpMenu.opened := true;
  end;
end;

procedure SetZforComponent(Element: TGUIObject);
begin
  if Element.used then
    Element._Z := Element.Z + Element.parent.zTemp;
end;

procedure TWindow.DoForEach(DoSomething: TDoSomething);
var
  i,j: integer;
begin
  for i := 0 to High(Buttons) do
    DoSomething(Buttons[i]);
  for i := 0 to High(Text) do
    DoSomething(Text[i]);
  for i := 0 to High(Checkboxes) do
    DoSomething(Checkboxes[i]);
  for i := 0 to High(ProgressBars) do
    DoSomething(ProgressBars[i]);
  for i := 0 to High(Panels) do
    DoSomething(Panels[i]);
  for i := 0 to High(Editfields) do
    DoSomething(Editfields[i]);
  for i := 0 to High(Edits) do
    DoSomething(Edits[i]);
  for i := 0 to High(TextFields) do
    DoSomething(TextFields[i]);
  for i := 0 to High(Images) do
    DoSomething(Images[i]);
  for i := 0 to High(RadioButtons) do
    DoSomething(RadioButtons[i]);

  for i := 0 to High(Frames) do
    for j := 0 to High(Frames[i].Frames) do
      Frames[i].Frames[j].Window.DoForEach(DoSomething);

  for i := 0 to High(Comboboxes) do
  begin
    DoSomething(Comboboxes[i]);
    for j := 0 to High(Comboboxes[i].ComboboxItems) do
      DoSomething(Comboboxes[i].ComboboxItems[j]);
  end;

  for i := 0 to High(PopUpMenus) do
  begin
```

```
    DoSomething(PopUpMenus[i]);
    for j := 0 to High(PopUpMenus[i].PopupMenuItems) do
      DoSomething(PopUpMenus[i].PopupMenuItems[j]);
  end;

  DoSomething(self);
  DoSomething(MinimizeButton);
  DoSomething(MaximizeButton);
  DoSomething(CloseButton);
end;

{#########################################################################
#################################}
Rendern der Quads...
#########################################################################
#################################}
procedure glRenderQuad(x1, y1, x2, y2: integer;  z: single); overload;
begin
  glBegin(GL_QUADS);
    glTexCoord2i(0,1); glVertex3f(x1,y1,z);
    glTexCoord2i(0,0); glVertex3f(x1,y2,z);
    glTexCoord2i(1,0); glVertex3f(x2,y2,z);
    glTexCoord2i(1,1); glVertex3f(x2,y1,z);
  glEnd;
end;

procedure glRenderQuad(x1, y1, x2, y2: integer; z, tx1, ty1, tx2, ty2: single); overload;
begin
  glBegin(GL_QUADS);
    glTexCoord2f(tx1,ty2); glVertex3f(x1,y1,z);
    glTexCoord2f(tx1,ty1); glVertex3f(x1,y2,z);
    glTexCoord2f(tx2,ty1); glVertex3f(x2,y2,z);
    glTexCoord2f(tx2,ty2); glVertex3f(x2,y1,z);
  glEnd;
end;

{#########################################################################
#################################}
Ob die Mouse auf einem Element ist...
#########################################################################
#################################}

function  TGUIObject.MouseonElement(mX,mY: integer): boolean;
begin
  result := false;
  if Visible then
    if ((parent.X + X)<=mX) and
       ((parent.Y + Y)<=mY) and
       ((parent.X + X + Width)>=mX) and
       ((parent.Y + Y + Height)>=mY) then
      result := true;
end;
```

```
function  TWindow.MouseonElement(mX,mY: integer): boolean;
begin
  result := false;
  if Visible then
    if MinimizeButton.minimized then
    begin
    if (X<=mX) and
       (Y<=mY) and
       ((X + Width)>=mX) and
       ((Y + CaptionBarHeight)>=mY) then
      result := true
    end
    else
    if (X<=mX) and
       (Y<=mY) and
       ((X + Width)>=mX) and
       ((Y + Height)>=mY) then
      result := true;
end;

function  TPopUpMenu.MouseonElement(mX,mY: integer): boolean;
begin
  result := false;
  if Visible and Opened then
    if (X<=mX) and
       (Y<=mY) and
       ((X + Width)>=mX) and
       ((Y + Height*length(PopUpMenuItems))>=mY) then
      result := true;
end;

function  TCombobox.MouseOnElement(mX,mY: integer): boolean;
begin
  if not opened then
  begin
    result := false;
    if Visible then
      if ((parent.X + X)<=mX) and
         ((parent.Y + Y)<=mY) and
         ((parent.X + X + Width)>=mX) and
         ((parent.Y + Y + Height)>=mY) then
        result := true;
  end else
  begin
    result := false;
    if Visible then
      if ((parent.X + X)<=mX) and
         ((parent.Y + Y)<=mY) and
         ((parent.X + X + Width)>=mX) and
         ((parent.Y + Y + space*(length(ComboboxItems))+height)>=mY) then
        result := true;
```

```
  end;
end;

function  TFrames.MouseOnElement(mX,mY: integer): boolean;
begin
  result := false;
  if Visible then
    if ((parent.X + X)<=mX) and
       ((parent.Y + Y)<=mY) and
       ((parent.X + X + Width)>=mX) and
       ((parent.Y + Y + CaptionBarHeight)>=mY) then
      result := true;
end;
{##################################################################
##################}
{##################################################################
##################}
{-------------  Add Proceduren der Komponenten -------------------------------}
{##################################################################
##################}
{##################################################################
##################}
{Tiefen der Objekte:
* Button: 1.1
* Text: 1.2
* checkbox: 0.9
* progressbar: 0.2
* panel: 0.1
* edit 1
* radiobutton: 0.8
}
function TWindow.AddButton(btnX, btnY, btnWidth, btnHeight : Integer; btnCaption : String):
integer;
begin
  if Buttons[High(Buttons)].used then
  begin
    setlength(Buttons, High(Buttons) + 2);
    Buttons[High(Buttons)]        := TButton.Create;
  end;

  with Buttons[High(Buttons)] do
  begin
    Parent :=  Pself;
    X :=btnx;
    Y :=btnY;
    Z := Parent.Z + 1.1;
    Width   :=btnWidth;
    Height  :=btnHeight;
    Caption :=btnCaption;
    used := true;
    Skin := self.skin;
    Font := self.font;
```

```
    Objecttype := 0;
    Index := High(Buttons);
    Visible := true;
    TextColor := Parent.FontColor;
    PopUpMenu := nil;
    plusx := 5;
    plusy := 5;
  end;
  result := High(Buttons);
end;

function TWindow.AddText(tX, tY : Integer; tText : String; tColor: TColor): integer;
begin
  if Text[High(Text)].used then
  begin
    setlength(Text, High(Text) + 2);
    Text[High(Text)]           := GUI.TText.Create;
  end;
  Text[High(Text)].Parent := PSelf;
  with Text[High(Text)] do
  begin
    X := tX;
    Y := tY;
    Z := Parent.Z + 1.2;
    Skin := self.skin;
    Font := self.font;
    text := ttext;
    Objecttype := 1;
    Index := High(self.Text);
    Visible := true;
    Color := tColor;
    used := true;
  end;
  result := High(Text);
end;

function TWindow.AddPanel(pX, pY, pWidth, pHeight : Integer): integer;
begin
  if Panels[High(Panels)].used then
  begin
    setlength(Panels, High(Panels) + 2);
    Panels[High(Panels)]          := TPanel.Create;
  end;
  Panels[High(Panels)].Parent :=Pself;
  with Panels[High(Panels)] do
  begin
    X :=pX;
    Y :=pY;
    Z := Parent.Z + 0.1;
    Width   :=pWidth;
    Height  :=pHeight;
    Visible :=true;
```

```
    Skin := self.skin;
    Font := self.font;
    Objecttype := 2;
    Index := High(Panels);
    used := true;
  end;
  result := High(Panels);
end;

function TWindow.AddProgressBar(pX, pY, pWidth, pHeight : Integer): integer;
begin
  if ProgressBars[High(ProgressBars)].used then
  begin
    setlength(ProgressBars, High(ProgressBars) + 2);
    ProgressBars[High(ProgressBars)]     := TProgressBar.Create;
  end;
  with ProgressBars[High(ProgressBars)] do
  begin
    Parent :=Pself;
    X :=pX;
    Y :=pY;
    Z := Parent.Z + 0.2;
    Width   :=pWidth;
    Height  :=pHeight;
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 3;
    Index := High(ProgressBars);
    used := true;
  end;
  result := High(ProgressBars);
end;

function TWindow.AddEdit(pX, pY, pWidth, pHeight : Integer; pText: String): integer;
begin
  if Edits[High(Edits)].used then
  begin
    setlength(Edits, High(Edits) + 2);
    Edits[High(Edits)]          := TEdit.Create;
  end;
  Edits[High(Edits)].Parent :=Pself;
  with Edits[High(Edits)] do
  begin
    X :=pX;
    Y :=pY;
    Z := Parent.Z + 1;
    Width   :=pWidth;
    Height  :=pHeight;
    Visible :=true;
    Text    :=pText;
    Skin := self.skin;
```

```
    Font := self.font;
    Objecttype := 4;
    Index := High(Edits);
    TextColor := cBlack;
    used := true;
    plusx:= 8;
    plusy:= 4;
  end;
  result := High(Edits);
end;

function TWindow.AddCheckbox(cbX, cbY, cbwidth, cbheight : Integer; cbChecked : Boolean):
integer;
begin
  if Checkboxes[High(Checkboxes)].used then
  begin
    setlength(Checkboxes, High(Checkboxes) + 2);
    Checkboxes[High(Checkboxes)]      := TCheckbox.Create;
  end;
  Checkboxes[High(Checkboxes)].Parent :=Pself;
  with Checkboxes[High(Checkboxes)] do
  begin
    X :=cbX;
    Y :=cbY;
    Z := Parent.Z + 0.9;
    Visible :=true;
    Checked :=cbChecked;
    pressed := false;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 5;
    Index := High(Checkboxes);
    width := cbwidth;
    height := cbheight;
    used := true;
  end;
  result := High(Checkboxes);
end;

function TWindow.AddRadioButton(rbX, rbY, rbwidth, rbheight, rbGroup : Integer; rbChecked :
Boolean): integer;
begin
  if RadioButtons[High(RadioButtons)].used then
  begin
    setlength(RadioButtons, High(RadioButtons) + 2);
    RadioButtons[High(RadioButtons)]    := TRadioButton.Create;
  end;
  RadioButtons[High(RadioButtons)].Parent :=Pself;
  with RadioButtons[High(RadioButtons)] do
  begin
    X :=rbX;
    Y :=rbY;
```

```
    Z := Parent.Z + 0.8;
    Checked :=rbChecked;
    width := rbwidth;
    height := rbHeight;
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 6;
    Index := High(RadioButtons);
    used := true;
    GroupIndex := AddRadioButtonToGroup(rbGroup, RadioButtons[High(RadioButtons)]);
  end;
  result := High(RadioButtons);
end;

function TWindow.AddRadioButtonToGroup(GroupIndex: integer; RadioButton: TRadioButton):
integer;
begin
  setlength(RadioButtonGroups[GroupIndex].RadioButtons,
High(RadioButtonGroups[GroupIndex].RadioButtons) + 2);
  RadioButtonGroups[GroupIndex].RadioButtons[High(RadioButtonGroups[GroupIndex].RadioButto
ns)] := RadioButton;
  RadioButton.Group := GroupIndex;
  RadioButton.ParentGroup := RadioButtonGroups[GroupIndex];
  RadioButton.GroupIndex := High(RadioButtonGroups[GroupIndex].RadioButtons);
  result:= High(RadioButtonGroups[GroupIndex].RadioButtons);
end;

function  TWindow.AddRadioButtonGroup: integer;
begin
  setlength(RadioButtonGroups, High(RadioButtonGroups) + 2);
  RadioButtonGroups[High(RadioButtonGroups)] := TRadioButtonGroup.Create;
  setlength(RadioButtonGroups[High(RadioButtonGroups)].RadioButtons, 1);
  RadioButtonGroups[High(RadioButtonGroups)].whichchecked := 0;
  result := High(RadioButtonGroups);
end;

function TWindow.AddChildWindow(pChild : TWindow): integer;
begin
  if haschilds then
    setlength(ChildWindows, High(ChildWindows) + 2);
  ChildWindows[High(ChildWindows)].Child := pChild;
  ChildWindows[High(ChildWindows)].Index := ChildWindows[High(ChildWindows)].Child.Index;
  ChildWindows[High(ChildWindows)].Child.Child := true;
  ChildWindows[High(ChildWindows)].used := true;
  hasChilds := true;
  result := High(ChildWindows);
end;

function  TWindow.AddImage(iX,iY,iWidth,iHeight: integer; iBlending: boolean; iSfactor, iDfactor:
Cardinal; igraphic: string): integer;
begin
```

```
  if Images[High(Images)].used then
  begin
    setlength(Images, High(Images) + 2);
    Images[High(Images)]    := TImage.Create;
  end;
  Images[High(Images)].Parent :=Pself;
  with Images[High(Images)] do
  begin
    X := iX;
    Y := iY;
    Z := Parent.Z + 0.13;
    color := cWhite;
    Width := iWidth;
    Height := iHeight;
    Blending := iBlending;
    sfactor := iSfactor;
    dfactor := iDfactor;
    LoadTexture(igraphic,graphic, false);
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 16;
    Index := High(Images);
    used := true;
    graphicpath := igraphic;
  end;
  result := High(Images);
end;

function  TWindow.AddPopupMenu(pWidth, pHeight: integer; firstCaption: string): integer;
begin
  if PopupMenus[High(PopupMenus)].used then
  begin
    setlength(PopupMenus, High(PopupMenus) + 2);
    PopupMenus[High(PopupMenus)]    := TPopupMenu.Create;
  end;
  PopupMenus[High(PopupMenus)].Parent :=Pself;
  with PopupMenus[High(PopupMenus)] do
  begin
    Index := High(PopupMenus);
    TextColor := parent.fontcolor;
    Z := Parent.Z + 1.25;
    opened := false;
    Width := pWidth;
    Height := pHeight;
    setlength(PopUpMenuItems, 1);
    PopUpMenuItems[0] := TPopUpMenuItem.Create;
    AddPopUpItem(firstCaption);
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 13;
```

```
    used := true;
    TextColor := cBlack;
    plusx:= 3;
    plusy:= 3;
  end;
  result := High(PopupMenus);
end;

function  TWindow.AddCombobox(cX,cY,cWidth,cHeight: integer; firstCaption: string): integer;
begin
  if Comboboxes[High(Comboboxes)].used then
  begin
    setlength(Comboboxes, High(Comboboxes) + 2);
    Comboboxes[High(Comboboxes)]    := TCombobox.Create;
  end;
  Comboboxes[High(Comboboxes)].Parent :=Pself;
  with Comboboxes[High(Comboboxes)] do
  begin
    X := cX;
    Y := cY;
    Z := Parent.Z + 0.7;
    Index := High(Comboboxes);
    TextColor := parent.fontcolor;
    Width := cWidth;
    Height := cHeight;
    setlength(ComboboxItems, 1);
    ComboboxItems[0] := TComboboxItem.Create;
    AddComboboxItem(firstCaption);
    Caption := firstCaption;
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 14;
    used := true;
    TextColor := cBlack;
    plusx:= 3;
    plusy:= 3;
    space := height;
  end;
  result := High(Comboboxes);
end;

function  TWindow.AddFrames(cX,cY,cWidth,cHeight: integer; firstWindow: TWindow): integer;
begin
  if Frames[High(Frames)].used then
  begin
    setlength(Frames, High(Frames) + 2);
    Frames[High(Frames)]    := TFrames.Create;
  end;
  Frames[High(Frames)].Parent :=Pself;
  with Frames[High(Frames)] do
  begin
```

```
    X := cX;
    Y := cY;
    Z := Parent.Z + 0.121;
    Width := cWidth;
    Height := cHeight;
    captionbarheight := 20;
    TextColor := cBlack;
    setlength(Frames,0);
    AddFrame(firstWindow);
    Visible :=true;
    Skin := self.skin;
    Font := self.font;
    Objecttype := 15;
    Index := High(Frames);
    used := true;
    tabwidth := 60;
    plusx := 4;
    plusy := 2;
  end;
  result := High(Frames);
end;

function  TWindow.AddTextField(tX,tY,tWidth,tHeight: integer; ttext: string; tfont: integer = -1):
integer;
begin
  if TextFields[High(TextFields)].used then
  begin
    setlength(TextFields, High(TextFields) + 2);
    TextFields[High(TextFields)]    := TTextField.Create;
  end;
  TextFields[High(TextFields)].Parent :=Pself;
  with TextFields[High(TextFields)] do
  begin
    X := tX;
    Y := tY;
    Z := Parent.Z + 0.14;
    Height := tHeight;
    Width := tWidth;
    font := tfont;
    Visible :=true;
    Skin := self.skin;
    if font = -1 then
      Font := self.font
    else
      Font := tfont;
    Objecttype := 12;
    Index := High(TextFields);
    used := true;
    TextColor := parent.fontcolor;
    space := Guiclass.fonts[font].Height - 4;
    text := ttext;
    TextColor := cBlack;
```

```
  end;
  result := High(TextFields);
end;

function  TWindow.AddEditField(tX,tY,tWidth,tHeight: integer; ttext: string; tfont: integer = -1):
integer;
begin
  if EditFields[High(EditFields)].used then
  begin
    setlength(EditFields, High(EditFields) + 2);
    EditFields[High(EditFields)]    := TEditField.Create;
  end;
  EditFields[High(EditFields)].Parent :=Pself;
  with EditFields[High(EditFields)] do
  begin
    X := tX;
    Y := tY;
    Z := Parent.Z + 0.15;
    Width := tWidth;
    Height := tHeight;
    font := tfont;
    text := ttext;
    Visible :=true;
    Skin := self.skin;
    if font = -1 then
      Font := self.font
    else
      Font := tfont;
    Objecttype := 11;
    Index := High(EditFields);
    used := true;
    TextColor := cBlack;
    space := Guiclass.fonts[font].Height - 4;
    plusx:= 5;
    plusy:= 5;
    line := 0;
    position := 0;
    setlength(Stringlist, 1);
  end;
  result := High(EditFields);
end;

{##########################################################################
###################}
{##########################################################################
###################}
{-------------- Render Proceduren der Komponenten ---------------------------}
{##########################################################################
###################}
{##########################################################################
###################}
```

```
procedure TGUIObject.RenderExact(X,Y,Width,Height: integer; Z, divi: single; plus: integer);
var
  X2,Y2:  integer;
begin
  X2 := X + Width;
  Y2 := Y + Height;

    //Render Edges
  glRenderQuad(X      , Y      , X+plus , Y+plus , Z,0     ,1-divi,divi  ,1      );    //oben links
  glRenderQuad(X      , Y2-plus, X+plus , Y2     , Z,0     ,0     ,divi  ,divi   );    //unten links
  glRenderQuad(X2-plus, Y2-plus, X2     , Y2     , Z,1-divi,0     ,1     ,divi   );    //unten rechts
  glRenderQuad(X2-plus, Y      , X2     , Y+plus , Z,1-divi,1-divi,1     ,1      );    //oben rechts

  //Render Kanten
  glRenderQuad(X+plus , Y      , X2-plus, Y+plus , Z,divi  ,1-divi,1-divi,1      );   //oben
  glRenderQuad(X      , Y+plus , X+plus , Y2-plus, Z,0     ,divi  ,divi  ,1-divi );    //links
  glRenderQuad(X+plus , Y2-plus, X2-plus, Y2     , Z,divi  ,0     ,1-divi,divi   );    //unten
  glRenderQuad(X2-plus, Y+plus , X2     , Y2-plus, Z,1-divi,divi  ,1     ,1-divi );    //rechts

  //Render Middle
  glRenderQuad(X+plus , Y+plus , X2-plus, Y2-plus, Z,divi  ,divi  ,1-divi,1-divi );          {}
end;

procedure TEditField.Render;
var
  i: integer;
begin
  glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].editfield);
  //glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
  RenderExact(parent.X + X,parent.Y + Y,Width,Height,Z, 1/4, 20);
  glColor4f(TextColor.r,TextColor.g,TextColor.b,Parent.Color.a);
  for i := 0 to High(Stringlist) do
    GUIclass.PrintFont(Parent.X + x + plusx,Parent.Y +
y+space*i+plusy,Z+0.01,font,PChar(Stringlist[i]));
  if active and GUIclass.showcursor then
    GUIclass.PrintFont(round(parent.X + X + plusx + position*GUIclass.Fonts[font].width -
GUIclass.Fonts[font].width/2),Parent.Y + y+space*line+plusy,Z+0.01,font, '|');
  glColor4f(Parent.Color.r,Parent.Color.g,Parent.Color.b,Parent.Color.a);
end;

procedure TImage.Render;
begin
  glBindTexture(GL_TEXTURE_2D, graphic);
  glColor4f(Color.r,Color.g,Color.b,parent.Color.a);
  if Blending then
    glBlendfunc(sfactor,dfactor)
  else
    glDisable(GL_BLEND);
  glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
  if Blending then
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  else
```

```
    glEnable(GL_BLEND);
  glColor4f(parent.Color.r,parent.Color.g,parent.Color.b,parent.Color.a);
end;

procedure TCombobox.Render;
var
  i: integer;
begin
  glBindTexture(GL_Texture_2D, GUIclass.skins[skin].Combobox);
  glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+height, parent.Y + y+height, Z,
0,0,0.25,1);
  glRenderQuad(parent.X + x+height, parent.Y + y, parent.X + x+width-height+1, parent.Y +
y+height, Z, 0.25,0,1,1);
  glBindTexture(GL_Texture_2D, GUIclass.skins[skin].Combobox_button);
  glRenderQuad(parent.X + x+width-height, parent.Y + y, parent.X + x+width, parent.Y +
y+height, Z+0.001);
  glColor4f(TextColor.r,TextColor.g,TextColor.b,parent.color.a);
  GUIClass.PrintFont(parent.X + x + plusx, parent.Y + y + plusy, Z+0.003, font,
Pchar(ComboboxItems[whichchecked].Caption));
  glColor4f(parent.color.r,parent.color.g,parent.color.b,parent.color.a);
  if opened then
  begin
    glBindTexture(GL_Texture_2D, GUIclass.skins[skin].ground);
    glRenderQuad(parent.X + x + plusx, parent.Y + y+height, parent.X + x+width-plusx, parent.Y
+ y+space*(length(ComboboxItems)+1), Z);
    glBindTexture(GL_Texture_2D, GUIclass.skins[skin].combo_ground);
    glRenderQuad(parent.X + x+2+plusx, parent.Y + y + height+space*(mouseon)+2, parent.X +
x+width-2-plusx, parent.Y + y+height+space*(mouseon+1)-2, Z+0.0001);
    glColor4f(TextColor.r,TextColor.g,TextColor.b,parent.color.a);
    for i := 0 to High(ComboboxItems) do
      GUIClass.PrintFont(parent.X + x + plusx + 4, parent.Y + y + space*(i+1) + plusy, Z+0.01,
font, PChar(ComboboxItems[i].Caption));
    glColor4f(parent.color.r,parent.color.g,parent.color.b,parent.color.a);
  end;
end;

procedure TTextField.Render;
var
  i: integer;
begin
  glColor4f(TextColor.r,TextColor.g,TextColor.b,Parent.Color.a);
  for i := 0 to High(Stringlist) do
    GUIclass.PrintFont(parent.x + x,parent.y + y+space*i,Z,font,PChar(Stringlist[i]));
  glColor4f(Parent.Color.r,Parent.Color.g,Parent.Color.b,Parent.Color.a);
end;

procedure TPopUpMenu.Render;
var
  i: integer;
begin
  _counter := High(PopupMenuItems);
  glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[skin].ground);
```

```
  glRenderQuad(x,y,x+width,y+(height*length(PopupMenuItems)),Z-0.02);
  glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[skin].combo_ground);
  glRenderQuad(x+2,y+height*whichchecked+2,x+width-2,y+height*(whichchecked+1)-2,Z-
0.01);
  glColor4f(TextColor.r,TextColor.g,TextColor.b,parent.Color.a);
  for i := 0 to High(PopUpMenuItems) do
    GUIclass.PrintFont(x+plusx,y+plusy+height*i,Z,font,PChar(PopUpMenuItems[i].Caption));
  glColor4f(parent.Color.r,parent.Color.g,parent.Color.b,parent.Color.a);
end;

procedure TFrames.Render;
var
  i: integer;
begin
  for i := 0 to High(Frames) do
    if i = whichchecked then
    begin
      glBindTexture(GL_TEXTURE_2D, Guiclass.skins[skin].frame_c);
      glRenderQuad(parent.X+X+tabwidth*i, parent.Y+Y, parent.X+X+tabwidth*(i+1),
parent.Y+Y+CaptionBarHeight, Z);
      glColor4f(TextColor.r,TextColor.g,TextColor.b,Parent.Color.a);
      Guiclass.PrintFont(parent.X+X+tabwidth*i+plusx, parent.Y+Y+plusy, Z+0.01,
font,PChar(Frames[i].Window.Caption));
      glColor4f(Parent.Color.r,Parent.Color.g,Parent.Color.b,Parent.Color.a);
    end else
    begin
      glBindTexture(GL_TEXTURE_2D, Guiclass.skins[skin].frame);
      glRenderQuad(parent.X+X+tabwidth*i, parent.Y+Y, parent.X+X+tabwidth*(i+1),
parent.Y+Y+CaptionBarHeight, Z);
      glColor4f(TextColor.r,TextColor.g,TextColor.b,Parent.Color.a);
      Guiclass.PrintFont(parent.X+X+tabwidth*i+plusx, parent.Y+Y+plusy, Z+0.01,
font,PChar(Frames[i].Window.Caption));
      glColor4f(Parent.Color.r,Parent.Color.g,Parent.Color.b,Parent.Color.a);
    end;
  Frames[whichchecked].Window.Render;
end;

procedure TButton.Render;
begin
  if pressed then
    glBindTexture(GL_TEXTURE_2D, GUIclass.skins[Skin].button_m)
  else
    if active then
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].button_c)
    else
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].button);

  //glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
  RenderExact(parent.X + X, parent.Y + y, width, height, Z, 1/4, 10);

  glColor4f(TextColor.r,TextColor.g,TextColor.b,parent.Color.A);
  GUIClass.PrintFont(parent.X+x+plusx, parent.Y+y+plusy, Z+0.01, font, Pchar(caption));
```

```
  glColor4f(parent.Color.r,parent.Color.g,parent.Color.b,parent.Color.A);
end;

procedure TPanel.Render;
begin
  glBindTexture(GL_TEXTURE_2D, GUIclass.skins[Skin].panel);
  //glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height,Z);
  RenderExact(parent.X + x, parent.Y + y, width, height,Z,1/4,15);
end;

procedure TProgressBar.Render;
begin
  glBindTexture(GL_TEXTURE_2D, GUIclass.skins[Skin].progress_ground);
  glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);

  glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].progressBar);
  glRenderQuad(parent.X + x+4, parent.Y + y+4, round(parent.X + x+(width*progress))-4,
parent.Y + y+height-4, Z+0.01, 0, 0, progress-0.04, 1);
end;

procedure TEdit.Render;
begin
  glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].Edit);
  RenderExact(parent.X + x, parent.Y + y, width, height, Z,1/4,8);
  //glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
  glColor4f(TextColor.r,TextColor.g,TextColor.b,parent.Color.A);
  GUIclass.PrintFont(parent.X + X + plusx,parent.Y + Y+plusy, Z+0.01,font,PChar(text));
  if active and GUIclass.showcursor then
    GUIclass.PrintFont(round(parent.X + X + plusx + Mouseposition*GUIclass.Fonts[font].width -
GUIclass.Fonts[font].width/2),parent.Y + y + plusy,Z+0.01,font, '|');
  glColor4f(parent.Color.r,parent.Color.g,parent.Color.b,parent.Color.A);
end;

procedure TCheckbox.Render;
begin
  if Checked then
    if pressed then
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].check_m_c)
    else
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].check_c)
  else
    if pressed then
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].check_m)
    else
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].check);

  glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
end;

procedure TRadioButton.Render;
begin
  if Checked then
```

```
    if pressed then
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].radio_m_c)
    else
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].radio_c)
  else
    if pressed then
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].radio_m)
    else
      glBindTexture(GL_TEXTURE_2D, GUIclass.skins[skin].radio);

  glRenderQuad(parent.X + x, parent.Y + y, parent.X + x+width, parent.Y + y+height, Z);
end;

{##################################################################################}
{##################################################################################}
{------------------------ Update der Elemente -----------------------------}
{##################################################################################}
{##################################################################################}

procedure TFrames.MoveFrame(dX,dY: integer);
var
  i: integer;
begin
  for i := 0 to High(Frames) do
  begin
    Frames[i].Window.X := Frames[i].Window.X + dX;
    Frames[i].Window.Y := Frames[i].Window.Y + dY;
  end;
end;

procedure TGUIField.Update(Value :string);
var
  i,j,k: integer;
  left: integer;
  place: integer;
  done: boolean;
  lastenter: integer;
begin
  setlength(Stringlist,0);
  if GUIclass.Fonts[Font].Width <> 0 then               //wegen division durch 0 -> keine exception
    place := trunc(Width / GUIclass.Fonts[Font].Width) -3
  else
  begin
  {$ifdef showerrors}
    showmessage('Clown! Don''t use Width := 0!');
  {$endif}
  end;
  j:=0;
```

```
  left:=1;
  lastenter := 0;
  if value <> '' then
  begin
    for i := 0 to length(value) - 1 do
    begin
      done := false;
      if Value[i] = chr(13) then
      begin
        setlength(Stringlist, High(Stringlist) + 2);
        stringlist[j] := copy(Value,left,i-left);
        left := i + 1;
        lastenter := i;
        inc(j);
      end;
      if ((i-lastenter) mod place) = (place - 1) then
      begin
        for k := i downto (i-place) do
          if not done and (value[k] = ' ') then
          begin
            setlength(Stringlist, High(Stringlist) + 2);
            stringlist[j] := copy(Value,left,k-left);
            left := k + 1;
            done := true;
            inc(j);
          end;
      end;
    end;
    if left <= (length(value) - 1) then
    begin
      setlength(Stringlist, High(Stringlist) + 2);
      stringlist[j] := copy(Value,left,length(value) - 1);
    end;
  end;
end;

function  TGUIField.GetText: string;
var
  i: integer;
begin
  for i := 0 to High(Stringlist) do
    result := result + Stringlist[i] + chr(13);
  result := copy(result, 0, length(result) - 1);  //muss gemacht werden weil am schluss kein Enter
mehr erfolgt.
end;

procedure TGUIField.MakeNewLine(which: integer);
var
  i: integer;
  s, s2: string;
begin
  setlength(stringlist, length(stringlist) + 1);
```

```
  s := '';
  for i := which to High(stringlist) do
  begin
    s2 := stringlist[i];
    stringlist[i] := s;
    s :=s2;
  end;
end;

procedure TGUIField.DeleteLine(which: integer);
var
  i: integer;
  s: string;
begin
  for i := which to High(Stringlist) - 1 do
    Stringlist[i] := Stringlist[i+1];
  setlength(Stringlist, High(Stringlist));
end;

function  TFrames.AddFrame(Window: TWindow): integer;
begin
  Window.Child := true;
  setlength(Frames, High(Frames) + 2);
  Frames[High(Frames)].Window := Window;
  Window.X := Parent.X + X;
  Window.Y := Parent.Y + Y + CaptionBarHeight;
  Window.Width := Width;
  Window.Height := Height - CaptionBarHeight;
  Window.CaptionBar := false;
  Window.Parent := Window;
  Window.isTab  := true;
  if High(Frames) > 0 then
    Window._Visible := false
  else
    Window.Visible := true;
  result := High(Frames);
  Window.Z := parent.Z + 1.5;
end;

function  TPopupMenu.AddPopUpItem(Caption: string): integer;
begin
  if PopUpMenuItems[High(PopUpMenuItems)].used then
  begin
    setlength(PopUpMenuItems, High(PopUpMenuItems) + 2);
    PopUpMenuItems[High(PopUpMenuItems)] := TPopUpMenuItem.Create;
  end;
  PopUpMenuItems[High(PopUpMenuItems)].used := true;
  PopUpMenuItems[High(PopUpMenuItems)].Index := High(PopUpMenuItems);
  PopUpMenuItems[High(PopUpMenuItems)].Caption := Caption;
  PopUpMenuItems[High(PopUpMenuItems)].Tag := Index;     //wir missbrauchen das x als parent
index...
  PopUpMenuItems[High(PopUpMenuItems)].objecttype := 30;
```

```
  PopUpMenuItems[High(PopUpMenuItems)].Z := Parent.Z + 1.25;
  PopUpMenuItems[High(PopUpMenuItems)].ParentMenu := Index;
  PopUpMenuItems[High(PopUpMenuItems)].mParent := self;
  PopUpMenuItems[High(PopUpMenuItems)].Parent := Parent;
  result := High(PopUpMenuItems);
end;

function  TCombobox.AddComboBoxItem(Caption: string): integer;
begin
  if ComboboxItems[High(ComboboxItems)].used then
  begin
    setlength(ComboboxItems, High(ComboboxItems) + 2);
    ComboboxItems[High(ComboboxItems)] := TComboboxItem.Create;
  end;
  ComboboxItems[High(ComboboxItems)].used := true;
  ComboboxItems[High(ComboboxItems)].Index := High(ComboboxItems);
  ComboboxItems[High(ComboboxItems)].Caption := Caption;
  ComboboxItems[High(ComboboxItems)].objecttype := 31;
  ComboboxItems[High(ComboboxItems)].Z := Parent.Z + 1.25;
  ComboboxItems[High(ComboboxItems)].Parent := Parent;
  result := High(ComboboxItems);
end;

procedure TText.Update(Value :string);
begin
  FText := Value;
  //länge und breite bestimmen um sagen können ob mousedown erfolgt ist
  width := round(length(text)*GUIclass.Fonts[font].width);
  height := GUIclass.Fonts[font].height;
end;

//das ändern von Events:
procedure TGUIObject.changeonmouseover(Value: boolean);
var
  b:  boolean;
  i:  integer;
begin
  b := false;
  Fonmouseoverevent := Value;
  if Value then Parent.hasonmouseover := true
  else
  begin
    if parent.onmouseoverevent then b := true;
    for i := 0 to High(parent.Buttons) do
      if parent.Buttons[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.Text) do
      if parent.Text[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.Checkboxes) do
      if parent.Checkboxes[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.ProgressBars) do
      if parent.ProgressBars[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.panels) do
```

```
      if parent.panels[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.edits) do
      if parent.edits[i].onmouseoverevent then b := true;
    for i := 0 to High(parent.radiobuttons) do
      if parent.radiobuttons[i].onmouseoverevent then b := true;
    Value := b;
  end;
end;

procedure TGUIObject.setonclick(Value: Tonclick);
begin
  onclickevent := true;
  Fonclick := Value;
  if not Assigned(Value) then onclickevent := false;
end;

procedure TGUIObject.setMousedown(Value: TonMousedown);
begin
  onMousedownevent := true;
  FonMousedown := Value;
  if not Assigned(Value) then onMousedownevent := false;
end;

procedure TGUIObject.setMouseup(Value: TonMouseUp);
begin
  onMouseUpEvent := true;
  FonMouseUp := Value;
  if not Assigned(Value) then onMouseUpEvent := false;
end;

procedure TGUIObject.setMouseover(Value: TonMouseOver);
begin
  onMouseoverEvent := true;
  FonMouseover := Value;
  if not Assigned(Value) then onMouseoverEvent := false;
end;

procedure TGUIObject.setkeydown(Value: TonKeydown);
begin
  onKeydownEvent := true;
  FonKeydown := Value;
  if not Assigned(Value) then onKeydownEvent := false;
end;

procedure TGUIObject.setkeyup(Value: TonKeyUp);
begin
  onKeyUpEvent := true;
  FonKeyUp := Value;
  if not Assigned(Value) then onKeyUpEvent := false;
end;

procedure TGUIObject.setafterdrag(Value: Tafterdrag);
```

```
begin
  afterdragEvent := true;
  Fafterdrag := Value;
  if not Assigned(Value) then afterdragEvent := false;
end;

procedure TGUIObject.setondrag(Value: TonDrag);
begin
  onDragEvent := true;
  Fondrag := Value;
  if not Assigned(Value) then onDragEvent := false;
end;

procedure TGUIObject.setonDestroy(Value: TonDestroy);
begin
  onDestroyEvent := true;
  FonDestroy := Value;
  if not Assigned(Value) then onDestroyEvent := false;
end;

procedure TGUIObject.setPopUpMenu(Value: TPopUpMenu);
begin
  FPopUpMenu := Value;
  if Value = nil then
  begin
    hasPopUpMenu := false;
    onMouseDown := nil;
  end else
  begin
    hasPopUpMenu := true;
    onMousedown := ShowPopUpMenu;
  end;
end;

procedure TGUIObject.setX(Value: integer);
var
  i,j: integer;
begin
  if (objecttype = 7) or (objecttype = 15) then
  begin
    for i := 0 to High(GUIclass.Windows[Index].Frames) do
      Parent.Frames[i].MoveFrame(Value-_X,0);
    Parent.ChangeCaptionBarButtons;
  end;
  _X := Value;
end;

procedure TGUIObject.setY(Value: integer);
var
  i: integer;
begin
  if (objecttype = 7) or (objecttype = 15) then
```

```
  begin
    for i := 0 to High(GUIclass.Windows[Index].Frames) do
      Parent.Frames[i].MoveFrame(0,Value-_Y);
    Parent.ChangeCaptionBarButtons;
  end;
  _Y := Value;
end;

procedure TGUIObject.setZ(Value: single);
var
  i: integer;
begin
  {if(objecttype = 7) or (objecttype = 50) or (objecttype = 30) or (objecttype = 31)then
  begin     }
  if(objecttype = 7) then
  begin
    GUIclass.Windows[index].zTemp := value-_Z;
    GUIclass.Windows[index].DoForEach(SetZforComponent);
  end;

  if objecttype = 15 then
  begin
    for i := 0 to High(Parent.Frames[Index].Frames) do
      Parent.Frames[Index].Frames[i].Window.Z := Value;
  end;
    {_Z := Value;
  end
    else}
  _Z := Value {+ Parent._Z};
end;

procedure TGUIObject.SetWidth(Value: integer);
var
  i: integer;
begin
  if objecttype = 15 then
    for i := 0 to High(Parent.Frames[Index].Frames) do
      Parent.Frames[Index].Frames[i].Window._Width := Value;
  _Width := Value;
end;

procedure TGUIObject.SetHeight(Value: integer);
var
  i: integer;
begin
  if objecttype = 15 then
    for i := 0 to High(Parent.Frames[Index].Frames) do
      Parent.Frames[Index].Frames[i].Window._Height := Value-
Parent.Frames[Index].CaptionBarHeight;
  _Height := Value;
end;
```

```
procedure TGUIObject.SetVisible(Value: boolean);
var
  i, j: integer;
begin
  if not Value then
  begin
    if OnDestroyEvent then
      onDestroy(self);
    if objecttype = 7 then
    begin
      Guiclass.WinOrder[Index] := -1;
      for i := 0 to High(Parent.Frames) do
        if Parent.Frames[i].used then
          for j := 0 to High(Parent.Frames[i].Frames) do
            Parent.Frames[i].Frames[j].Window.Visible := false;
      for i := 0 to High(Parent.ChildWindows) do
        if Parent.ChildWindows[i].used then
          Parent.ChildWindows[i].Child.Visible := false;
    end;
  end
  else
    if objecttype = 7 then
    begin
      Guiclass.WinOrder[Index] := -2;
      for i := 0 to High(Parent.Frames) do
        if Parent.Frames[i].used then
          Parent.Frames[i].Frames[0].Window.Visible := true;
      Guiclass.SetZforWindows;
    end;

  _Visible := Value;
end;

procedure TWindow.ChangeCaptionBarButtons;
begin
  with minimizeButton do
  begin
    onclick := minimize;
    Parent := PSelf;
    X := Parent.Width - (CaptionBarHeight*3) + 1 - 4;
    Y := 6;
    //Z := Parent.Z + 1.45;
    Width := CaptionBarHeight - 2;
    Height := CaptionBarHeight - 10;
    Objecttype := 8;
  end;
  with maximizeButton do
  begin
    onclick := ChangeWindowState;
    Parent := PSelf;
    X := Parent.Width - (CaptionBarHeight*2) + 1 - 4;
    Y := 6;
```

```
    //Z := Parent.Z + 1.45;
    Width := CaptionBarHeight - 2;
    Height := CaptionBarHeight - 10;
    Objecttype := 9;
  end;
  with closeButton do
  begin
    onclick := CloseWindow;
    Parent := PSelf;
    X := Parent.Width - CaptionBarHeight + 1 - 4;
    Y := 6;
    //Z := Parent.Z + 1.45;
    Width := CaptionBarHeight - 2;
    Height := CaptionBarHeight - 10;
    Objecttype := 10;
  end;
end;

{##################################################################
##################}
{##################################################################
##################}
{---------------- GUI Proceduren ----------------------------------------}
{##################################################################
##################}
{##################################################################
##################}
function TGUIclass.GetKey(Value: Word): String;
begin
{$IFDEF sdl}
  result := '';
  if Shift then
    case Value of
       32: result := ' ';
       48: result := '=';
       49: result := '!';
       50: result := '"';
       51: result := '§';
       52: result := '$';
       53: result := '%';
       54: result := '&';
       55: result := '/';
       56: result := '(';
       57: result := ')';
       97: result := 'A';
       98: result := 'B';
       99: result := 'C';
      100: result := 'D';
      101: result := 'E';
      102: result := 'F';
      103: result := 'G';
      104: result := 'H';
```

```
    105: result := 'I';
    106: result := 'J';
    107: result := 'K';
    108: result := 'L';
    109: result := 'M';
    110: result := 'N';
    111: result := 'O';
    112: result := 'P';
    113: result := 'Q';
    114: result := 'R';
    115: result := 'S';
    116: result := 'T';
    117: result := 'U';
    118: result := 'V';
    119: result := 'W';
    120: result := 'X';
    121: result := 'Z';
    122: result := 'Y';
     91: result := 'Ü';
     93: result := '*';
     44: result := ';';
     47: result := '_';
     46: result := ':';
     92: result := '''';
     59: result := 'Ö';
     45: result := '?';
     96: result := '°';
     61: result := '`';
     39: result := 'Ä';
     60: result := '>';
  end
else
if Alt then
  case Value of
     32: result := ' ';
     48: result := '}';
     49: result := '1';
     50: result := '²';
     51: result := '³';
     52: result := '4';
     53: result := '5';
     54: result := '6';
     55: result := '{';
     56: result := '[';
     57: result := ']';
     97: result := 'a';
     98: result := 'b';
     99: result := 'c';
    100: result := 'd';
    101: result := '€';
    102: result := 'f';
    103: result := 'g';
```

```
    104: result := 'h';
    105: result := 'i';
    106: result := 'j';
    107: result := 'k';
    108: result := 'l';
    109: result := 'µ';
    110: result := 'n';
    111: result := 'o';
    112: result := 'p';
    113: result := '@';
    114: result := 'r';
    115: result := 's';
    116: result := 't';
    117: result := 'u';
    118: result := 'v';
    119: result := 'w';
    120: result := 'x';
    121: result := 'z';
    122: result := 'y';
     91: result := 'ü';
     93: result := '~';
     44: result := ',';
     47: result := '-';
     46: result := '.';
     92: result := '#';
     59: result := 'ö';
     45: result := '\';
     96: result := '^';
     61: result := '´';
     39: result := 'ä';
     60: result := '|';
  end
 else
  case Value of
    32:  result := ' ';
    48:  result := '0';
    49:  result := '1';
    50:  result := '2';
    51:  result := '3';
    52:  result := '4';
    53:  result := '5';
    54:  result := '6';
    55:  result := '7';
    56:  result := '8';
    57:  result := '9';
    97:  result := 'a';
    98:  result := 'b';
    99:  result := 'c';
   100:  result := 'd';
   101:  result := 'e';
   102:  result := 'f';
   103:  result := 'g';
```

```
    104:  result := 'h';
    105:  result := 'i';
    106:  result := 'j';
    107:  result := 'k';
    108:  result := 'l';
    109:  result := 'm';
    110:  result := 'n';
    111:  result := 'o';
    112:  result := 'p';
    113:  result := 'q';
    114:  result := 'r';
    115:  result := 's';
    116:  result := 't';
    117:  result := 'u';
    118:  result := 'v';
    119:  result := 'w';
    120:  result := 'x';
    121:  result := 'z';
    122:  result := 'y';
     91:  result := 'ü';
     93:  result := '+';
     44:  result := ',';
     47:  result := '-';
     46:  result := '.';
     92:  result := '#';
     59:  result := 'ö';
     45:  result := 'ß';
     96:  result := '^';
     61:  result := '´';
     39:  result := 'ä';
     60:  result := '<';
   end;
{$ELSE}
  result := '';
  if Shift then
    case Value of
     32:  result := ' ';
     48:  result := '=';
     49:  result := '!';
     50:  result := '"';
     51:  result := '§';
     52:  result := '$';
     53:  result := '%';
     54:  result := '&';
     55:  result := '/';
     56:  result := '(';
     57:  result := ')';
     65:  result := 'A';
     66:  result := 'B';
     67:  result := 'C';
     68:  result := 'D';
     69:  result := 'E';
```

```
    70:  result := 'F';
    71:  result := 'G';
    72:  result := 'H';
    73:  result := 'I';
    74:  result := 'J';
    75:  result := 'K';
    76:  result := 'L';
    77:  result := 'M';
    78:  result := 'N';
    79:  result := 'O';
    80:  result := 'P';
    81:  result := 'Q';
    82:  result := 'R';
    83:  result := 'S';
    84:  result := 'T';
    85:  result := 'U';
    86:  result := 'V';
    87:  result := 'W';
    88:  result := 'X';
    89:  result := 'Y';
    90:  result := 'Z';
    186: result := 'Ü';
    187: result := '*';
    188: result := ';';
    189: result := '_';
    190: result := ':';
    191: result := '"';
    192: result := 'Ö';
    219: result := '?';
    220: result := '°';
    221: result := '`';
    222: result := 'Ä';
    226: result := '>';
  end
else
if Alt then
  case Value of
    32:  result := ' ';
    48:  result := '}';
    49:  result := '1';
    50:  result := '²';
    51:  result := '³';
    52:  result := '4';
    53:  result := '5';
    54:  result := '6';
    55:  result := '{';
    56:  result := '[';
    57:  result := ']';
    65:  result := 'a';
    66:  result := 'b';
    67:  result := 'c';
    68:  result := 'd';
```

```
    69:  result := '€';
    70:  result := 'f';
    71:  result := 'g';
    72:  result := 'h';
    73:  result := 'i';
    74:  result := 'j';
    75:  result := 'k';
    76:  result := 'l';
    77:  result := 'µ';
    78:  result := 'n';
    79:  result := 'o';
    80:  result := 'p';
    81:  result := '@';
    82:  result := 'r';
    83:  result := 's';
    84:  result := 't';
    85:  result := 'u';
    86:  result := 'v';
    87:  result := 'w';
    88:  result := 'x';
    89:  result := 'y';
    90:  result := 'z';
    186: result := 'ü';
    187: result := '~';
    188: result := ',';
    189: result := '-';
    190: result := '.';
    191: result := '#';
    192: result := 'ö';
    219: result := '\';
    220: result := '^';
    221: result := '´';
    222: result := 'ä';
    226: result := '|';
  end
else
  case Value of
    32:  result := ' ';
    48:  result := '0';
    49:  result := '1';
    50:  result := '2';
    51:  result := '3';
    52:  result := '4';
    53:  result := '5';
    54:  result := '6';
    55:  result := '7';
    56:  result := '8';
    57:  result := '9';
    65:  result := 'a';
    66:  result := 'b';
    67:  result := 'c';
    68:  result := 'd';
```

```
      69:  result := 'e';
      70:  result := 'f';
      71:  result := 'g';
      72:  result := 'h';
      73:  result := 'i';
      74:  result := 'j';
      75:  result := 'k';
      76:  result := 'l';
      77:  result := 'm';
      78:  result := 'n';
      79:  result := 'o';
      80:  result := 'p';
      81:  result := 'q';
      82:  result := 'r';
      83:  result := 's';
      84:  result := 't';
      85:  result := 'u';
      86:  result := 'v';
      87:  result := 'w';
      88:  result := 'x';
      89:  result := 'y';
      90:  result := 'z';
      186: result := 'ü';
      187: result := '+';
      188: result := ',';
      189: result := '-';
      190: result := '.';
      191: result := '#';
      192: result := 'ö';
      219: result := 'ß';
      220: result := '^';
      221: result := '´';
      222: result := 'ä';
      226: result := '<';
    end;
{$ENDIF}
end;

procedure TGUIclass.PrintTextarray;
var
  i: integer;
begin
  //glDepthMask(true);
  for i:= 0 to High(Text) do
    if Text[i].Visible then
      with Text[i] do
      begin
        glColor4d(color.r, color.g, color.b, color.a);
        PrintFont(x, y, z, font, Pchar(text));
      end;
  glColor4d(1,1,1,1);
  //glDepthMask(false);
```

```
end;

procedure TGUIclass.PrintFont(x, y: Integer; z: single; font: Integer; text : pchar);
begin
  glDepthMask(false);
  glBindTexture(GL_TEXTURE_2D, fonts[font].graphic);
  glPushMatrix();
    glTranslatef(x, y, z);
    glPushAttrib(GL_LIST_BIT);
      glListBase(fonts[font].list);
      glCallLists(length(text), GL_UNSIGNED_BYTE, text); // Write The Text To The Screen
    glpopattrib();
  glPopMatrix();
  glDepthMask(true);
end;

procedure TGUIclass.SetZforWindows;
var
  i,j, c2, number: integer;
  intarr: array of integer;
begin
  c2 := 0;

  //set the new windows in the right position.
  for i := 0 to High(Windows) do
    if Windows[i].Visible and not Windows[i].isTab and (Windows[i].Z > -1) and (Windows[i].Z <
31) then
    begin
      if Windows[i] = lastMouseDown.Parent then
      begin
        WinOrder[i] := 11;
      end else
      begin
        if WinOrder[i] = -2 then
        begin
          WinOrder[i] := c2;
          inc(c2);
        end;
      end;
    end
    else
      WinOrder[i] := -1;

  setlength(intarr, length(Winorder));
  //detect the order of the windows
  for i := 0 to High(Windows) do
    if WinOrder[i] >= 0 then
    begin
      number := 10;
      for j := 0 to High(Windows) do
        if (WinOrder[j] >= 0) and (i<>j) then
           if WinOrder[j] > WinOrder[i] then
```

```
          dec(number);
      intarr[i] := number;
    end
    else
      intarr[i] := -1;

  //copy the temp array to the right one
  for i := 0 to High(Winorder) do
    Winorder[i] := intarr[i];

  //set Z for each Winorder.
  for i := 0 to High(Windows) do
    if Winorder[i] >= 0 then
    begin
      Windows[i].Z := Winorder[i]*3;
      for j := 0 to High(Windows[i].Frames) do
        Windows[i].Frames[j].Z := Windows[i].Z + 1.5;
    end;
end;

procedure TGUIclass.MouseMove(newX,newY: integer);
{
*0:    Buttons
*1:    Text
*2:    Panels
*3:    ProgressBars
*4:    Edits
*5:    Checkboxes
*6:    RadioButtons
*7:    Windows
*8:    minimizeButton
*9:    maximizeButton
*10:   closeButton
}
var
  i,j: integer;
  newmouseon: single;
begin
  lastMouseMove := nil;
  if dragevent then
  begin
    if lastMouseDown.objecttype = 7 then
    begin
      if windows[lastMouseDown.Index].MaximizeButton.maximized = false then
      begin
        lastMouseDown.X := lastMouseDown.X + newX - Mouseold.x;
        lastMouseDown.Y := lastMouseDown.Y + newY - Mouseold.y;
        if lastMousedown.onDragEvent then lastMousedown.onDrag(newX, newY, lastMouseDown,
1);
        //for i := 0 to High(Windows[lastMouseDown.Index].Frames) do
          //Windows[lastMouseDown.Index].Frames[i].MoveFrame(newX - Mouseold.x,newY -
Mouseold.y);
```

```
    end;
  end else
  begin
    lastMouseDown.X := lastMouseDown.X + newX - Mouseold.x;
    lastMouseDown.Y := lastMouseDown.Y + newY - Mouseold.y;
    if lastMouseDown.onDragEvent then lastMouseMove.onDrag(newX, newY, lastMouseDown, 1);
    //if (lastMouseDown.objecttype = 15) and lastMouseDown.used then
      //lastMouseDown.Parent.Frames[lastMouseDown.Index].MoveFrame(lastMouseDown.X +
newX - Mouseold.x,lastMouseDown.Y + newY - Mouseold.y);
  end;
 end;

 for i := 0 to High(Windows) do
 begin
  if Windows[i].hasonmouseover then
    if Windows[i].MouseonElement(newX, newY) then
    begin
      if lastmouseMove = nil then
        lastmouseMove  := Windows[i]
      else
        if lastmouseMove.Z < Windows[j].Z then
          lastmouseMove  := Windows[j];

      //Buttons
      for j := 0 to High(Windows[i].Buttons) do
        if Windows[i].Buttons[j].onmouseoverevent then
          if Windows[i].Buttons[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Buttons[j].z then
              lastMouseMove := Windows[i].Buttons[j];

      //Text
      for j := 0 to High(Windows[i].Text) do
        if Windows[i].Text[j].onmouseoverevent then
          if Windows[i].Text[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Text[j].z then
              lastMouseMove := Windows[i].Text[j];

      //Panels
      for j := 0 to High(Windows[i].Panels) do
        if Windows[i].Panels[j].onmouseoverevent then
          if Windows[i].Panels[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Panels[j].z then
              lastMouseMove := Windows[i].Panels[j];

      //RadioButtons
      for j := 0 to High(Windows[i].RadioButtons) do
        if Windows[i].RadioButtons[j].onmouseoverevent then
          if Windows[i].RadioButtons[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].RadioButtons[j].z then
              lastMouseMove := Windows[i].RadioButtons[j];

      //Checkboxes
```

```
      for j := 0 to High(Windows[i].Checkboxes) do
        if Windows[i].Checkboxes[j].onmouseoverevent then
          if Windows[i].Checkboxes[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Checkboxes[j].z then
              lastMouseMove := Windows[i].Checkboxes[j];

      //ProgressBars
      for j := 0 to High(Windows[i].ProgressBars) do
        if Windows[i].ProgressBars[j].onmouseoverevent then
          if Windows[i].ProgressBars[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].ProgressBars[j].z then
              lastMouseMove := Windows[i].ProgressBars[j];

      //Edits
      for j := 0 to High(Windows[i].Edits) do
        if Windows[i].Edits[j].onmouseoverevent then
          if Windows[i].Edits[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Edits[j].z then
              lastMouseMove := Windows[i].Edits[j];

      //PopUpMenus
      {for j := 0 to High(Windows[i].PopUpMenus) do
        if Windows[i].PopUpMenus[j].onmouseoverevent and Windows[i].PopUpMenus[j].opened
then
          if Windows[i].PopUpMenus[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].PopUpMenus[j].z then
              lastMouseMove := Windows[i].PopUpMenus[j];   }

      //Comboboxes
      {for j := 0 to High(Windows[i].Comboboxes) do
        if Windows[i].Comboboxes[j].onmouseoverevent and Windows[i].Comboboxes[j].opened
then
          if Windows[i].Comboboxes[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Comboboxes[j].z then
              lastMouseMove := Windows[i].Comboboxes[j];    }

      //Frames
      for j := 0 to High(Windows[i].Frames) do
        if Windows[i].Frames[j].onmouseoverevent then
          if Windows[i].Frames[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Frames[j].z then
              lastMouseMove := Windows[i].Frames[j];

      //Images
      for j := 0 to High(Windows[i].Images) do
        if Windows[i].Images[j].onmouseoverevent then
          if Windows[i].Images[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].Images[j].z then
              lastMouseMove := Windows[i].Images[j];

      //EditFields
      for j := 0 to High(Windows[i].EditFields) do
```

```
        if Windows[i].EditFields[j].onmouseoverevent then
          if Windows[i].EditFields[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].EditFields[j].z then
              lastMouseMove := Windows[i].EditFields[j];

      //TextFields
      for j := 0 to High(Windows[i].TextFields) do
        if Windows[i].TextFields[j].onmouseoverevent then
          if Windows[i].TextFields[j].MouseonElement(newX, newY) then
            if lastMouseMove.Z < Windows[i].TextFields[j].z then
              lastMouseMove := Windows[i].TextFields[j];

    end;

  //Comboboxes
  for j := 0 to High(Windows[i].Comboboxes) do
    if Windows[i].Comboboxes[j].MouseonElement(newX,newY) then
    begin
      if lastMouseMove = nil then
      begin
        if not Windows[i].Comboboxes[j].opened then
          lastMouseMove := Windows[i].Comboboxes[j]
        else
        begin
          newmouseon := (newY - (Windows[i].Comboboxes[j].parent.y +
Windows[i].Comboboxes[j].Y +
Windows[i].Comboboxes[j].Height))/Windows[i].Comboboxes[j].space;
          if newmouseon < 0 then
            lastMouseMove := Windows[i].Comboboxes[j]
          else
          begin
            Windows[i].Comboboxes[j].mouseon := trunc(newmouseon);
            if newmouseon >= length(Windows[i].Comboboxes[j].ComboboxItems) then
              dec(Windows[i].Comboboxes[j].mouseon);
            lastMouseMove :=
Windows[i].Comboboxes[j].ComboboxItems[Windows[i].Comboboxes[j].mouseon];
          end;
        end;
      end else
        if lastMouseMove.Z < Windows[i].Comboboxes[j].Z then
          if not Windows[i].Comboboxes[j].opened then
            lastMouseMove := Windows[i].Comboboxes[j]
          else
          begin
            newmouseon := (newY - (Windows[i].Comboboxes[j].parent.y +
Windows[i].Comboboxes[j].Y +
Windows[i].Comboboxes[j].Height))/Windows[i].Comboboxes[j].space;
            if newmouseon < 0 then
              lastMouseMove := Windows[i].Comboboxes[j]
            else
            begin
              Windows[i].Comboboxes[j].mouseon := trunc(newmouseon);
```

```
            if newmouseon >= length(Windows[i].Comboboxes[j].ComboboxItems) then
              dec(Windows[i].Comboboxes[j].mouseon);
            lastMouseMove :=
Windows[i].Comboboxes[j].ComboboxItems[Windows[i].Comboboxes[j].mouseon];
          end;
        end;
    end;

   //PopUpMenus
   for j := 0 to High(Windows[i].PopupMenus) do
     if Windows[i].PopupMenus[j].opened then
       if Windows[i].PopupMenus[j].MouseonElement(newX,newY) then
       begin
         if lastMouseMove = nil then
         begin
           Windows[i].PopupMenus[j].whichchecked := trunc((newY -
Windows[i].PopupMenus[j].Y)/Windows[i].PopupMenus[j].Height);
           if Windows[i].PopupMenus[j].whichchecked =
length(Windows[i].PopupMenus[j].PopUpMenuItems) then
             dec(Windows[i].PopupMenus[j].whichchecked);
           lastMouseMove :=
Windows[i].PopupMenus[j].PopUpMenuItems[Windows[i].PopupMenus[j].whichchecked];
         end else
           if lastMouseMove.Z < Windows[i].PopupMenus[j].Z then
           begin
             Windows[i].PopupMenus[j].whichchecked := trunc((newY -
Windows[i].PopupMenus[j].Y)/Windows[i].PopupMenus[j].Height);
             if Windows[i].PopupMenus[j].whichchecked =
length(Windows[i].PopupMenus[j].PopUpMenuItems) then
               dec(Windows[i].PopupMenus[j].whichchecked);
             lastMouseMove :=
Windows[i].PopupMenus[j].PopUpMenuItems[Windows[i].PopupMenus[j].whichchecked];
           end;
       end;

  end;
  if (lastMouseMove <> nil) and lastMouseMove.onmouseoverevent then
    lastMouseMove.onmouseover(newX,newY,lastMouseMove);
  Mouseold.X := newX;
  Mouseold.Y := newY;
end;

procedure TGUIclass.MouseDown(mX,mY, mButton: integer);
var
  i, j, k: integer;
begin
  lastmousedown := nil;
  // "Checken" wo der Mausklick hin ist...
  for j:=0 to High(Windows) do
  begin
    if Windows[j].MouseonElement(mx,my) then
    begin
```

```
if lastmousedown = nil then
  lastMouseDown  := Windows[j]
else
  if lastMouseDown.Z < Windows[j].Z then
    lastMouseDown  := Windows[j];


// ***die einzelnen Elemente berechnen***
//panels
for i :=0 to High(Windows[j].Panels) do
  if Windows[j].Panels[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].Panels[i].Z then
      lastMouseDown  := Windows[j].Panels[i];

//ProgressBars
for i :=0 to High(Windows[j].ProgressBars) do
  if Windows[j].ProgressBars[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].ProgressBars[i].Z then
      lastMouseDown  := Windows[j].ProgressBars[i];

//Buttons
for i :=0 to High(Windows[j].Buttons) do
  if Windows[j].Buttons[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].Buttons[i].Z then
      lastMouseDown  := Windows[j].Buttons[i];

//Checkboxes
for i :=0 to High(Windows[j].Checkboxes) do
  if Windows[j].Checkboxes[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].Checkboxes[i].Z then
      lastMouseDown  := Windows[j].Checkboxes[i];

//RadioButtons
for i :=0 to High(Windows[j].RadioButtons) do
  if Windows[j].RadioButtons[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].RadioButtons[i].Z then
      lastMouseDown  := Windows[j].RadioButtons[i];

//Text
for i :=0 to High(Windows[j].Text) do
  if Windows[j].Text[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].Text[i].Z then
      lastMouseDown  := Windows[j].Text[i];

//Edits
for i :=0 to High(Windows[j].Edits) do
  if Windows[j].Edits[i].MouseonElement(mX, mY) then
    if lastMouseDown.Z < Windows[j].Edits[i].Z then
      lastMouseDown  := Windows[j].Edits[i];

//Frames
for i :=0 to High(Windows[j].Frames) do
```

```
      if Windows[j].Frames[i].MouseonElement(mX, mY) then
        if lastMouseDown.Z < Windows[j].Frames[i].Z then
          lastMouseDown  := Windows[j].Frames[i];

    //Images
    for i :=0 to High(Windows[j].Images) do
      if Windows[j].Images[i].MouseonElement(mX, mY) then
        if lastMouseDown.Z < Windows[j].Images[i].Z then
          lastMouseDown  := Windows[j].Images[i];

    //TextFields
    for i :=0 to High(Windows[j].TextFields) do
      if Windows[j].TextFields[i].MouseonElement(mX, mY) then
        if lastMouseDown.Z < Windows[j].TextFields[i].Z then
          lastMouseDown  := Windows[j].TextFields[i];

    //EditFields
    for i :=0 to High(Windows[j].EditFields) do
      if Windows[j].EditFields[i].MouseonElement(mX, mY) then
        if lastMouseDown.Z < Windows[j].EditFields[i].Z then
          lastMouseDown  := Windows[j].EditFields[i];

    {//Comboboxes
    for i :=0 to High(Windows[j].Comboboxes) do
      if Windows[j].Comboboxes[i].MouseonElement(mX, mY) then
        if lastMouseDown.Z < Windows[j].Comboboxes[i].Z then
          lastMouseDown  := Windows[j].Comboboxes[i]; }

    if windows[j].CaptionBar then
      if (((Windows[j].Y+Windows[j].CaptionBarHeight)>=mY)) and (mbutton = 1) then
        if lastMouseDown.Z < Windows[j].MinimizeButton.Z then                    //minibutton.z, weil
alle buttons gleiches z haben.
        begin
          if Windows[j].MinimizeButton.MouseonElement(mX, mY) then
            lastMouseDown  := Windows[j].MinimizeButton
          else
          if Windows[j].MaximizeButton.MouseonElement(mX, mY) then
            lastMouseDown  := Windows[j].MaximizeButton
          else
          if Windows[j].CloseButton.MouseonElement(mX, mY) then
            lastMouseDown  := Windows[j].CloseButton;
        end;
  end;

  //Comboboxes
  for i :=0 to High(Windows[j].Comboboxes) do
    if Windows[j].Comboboxes[i].MouseonElement(mX, mY) and Windows[j].Visible then
    begin
      if lastMouseDown = nil then
      begin
        if Windows[j].Comboboxes[i].opened then
        begin
```

```
          if (Windows[j].y + Windows[j].Comboboxes[i].y + Windows[j].Comboboxes[i].Height) >
my then
          begin
            lastMouseDown := windows[j].Comboboxes[i];
            Windows[j].Comboboxes[i].opened := false;
          end else
            lastMouseDown  :=
Windows[j].Comboboxes[i].ComboboxItems[Windows[j].Comboboxes[i].mouseon];
        end else
        begin
          lastMouseDown  := Windows[j].Comboboxes[i];
          Windows[j].Comboboxes[i].opened := true;
        end;
      end
      else
        if lastMouseDown.Z < Windows[j].Comboboxes[i].Z then
          if Windows[j].Comboboxes[i].opened then
          begin
            if (Windows[j].y + Windows[j].Comboboxes[i].y + Windows[j].Comboboxes[i].Height) >
my then
            begin
              lastMouseDown := windows[j].Comboboxes[i];
              Windows[j].Comboboxes[i].opened := false;
            end else
              lastMouseDown  :=
Windows[j].Comboboxes[i].ComboboxItems[Windows[j].Comboboxes[i].mouseon];
          end else
          begin
            lastMouseDown  := Windows[j].Comboboxes[i];
            Windows[j].Comboboxes[i].opened := true;
          end;
    end
    else
      Windows[j].Comboboxes[i].opened := false;


  //PopUpMenus
  for i :=0 to High(Windows[j].PopUpMenus) do
    if Windows[j].PopUpMenus[i].MouseonElement(mX, mY) and Windows[j].Visible then
    begin
      if lastMouseDown = nil then
      begin
        lastMouseDown  :=
Windows[j].PopupMenus[i].PopUpMenuItems[Windows[j].PopupMenus[i].whichchecked];
      end else
        if lastMouseDown.Z < Windows[j].PopupMenus[i].Z then
        begin
          lastMouseDown  :=
Windows[j].PopupMenus[i].PopUpMenuItems[Windows[j].PopupMenus[i].whichchecked];
        end;
    end else
      Windows[j].PopUpMenus[i].opened := false;
```

```
  end;

  //Die Auswertung des Klicks...
  if lastMouseDown <> nil then
  begin
    if lastMouseDown.dragevent and (mbutton = 1) then
    begin
      if lastMouseDown.objecttype = 7 then
      begin
        if windows[lastMouseDown.index].CaptionBar then
          if
((Windows[lastMouseDown.index].Y+Windows[lastMouseDown.index].CaptionBarHeight)>=mY)
then
            dragevent := true;
      end else
        dragevent := true;
      Mouseold.x := mx;
      Mouseold.y := my;
    end;

    case lastMouseDown.objecttype of
      0:  if mbutton = 1 then lastMouseDown.parent.Buttons[lastMouseDown.index].Pressed :=
true;
      4:  if (mButton = 1) then
            if (mx <> lastMouseDown.parent.Edits[lastMouseDown.index].X) then
            begin
              lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition   := round((mX -
(lastMouseDown.parent.Edits[lastMouseDown.index].X+lastMouseDown.parent.X +
lastMouseDown.parent.Edits[lastMouseDown.index].plusx))/Fonts[lastMouseDown.parent.Edits[las
tMouseDown.index].Font].Width);
              if lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition < 0 then
lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition := 0;
              if (length(lastMouseDown.parent.Edits[lastMouseDown.index].Text)) <
lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition then
                lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition :=
length(lastMouseDown.parent.Edits[lastMouseDown.index].Text)
            end else lastMouseDown.parent.Edits[lastMouseDown.index].MousePosition := 0;
      5:  if mbutton = 1 then lastMouseDown.parent.Checkboxes[lastMouseDown.index].Pressed :=
true;
      6:  if mbutton = 1 then lastMouseDown.parent.RadioButtons[lastMouseDown.index].Pressed
:= true;
      8:  lastMouseDown.parent.MinimizeButton.pressed := true;
      9:  lastMouseDown.parent.MaximizeButton.pressed := true;
      10: lastMouseDown.parent.CloseButton.pressed := true;
      11: if (mButton = 1) then
          begin
            if (mx <> lastMouseDown.parent.EditFields[lastMouseDown.index].X) then
            begin
              lastMouseDown.parent.EditFields[lastMouseDown.index].Position   := round((mX -
(lastMouseDown.parent.EditFields[lastMouseDown.index].X+lastMouseDown.parent.X +
lastMouseDown.parent.EditFields[lastMouseDown.index].plusx))/Fonts[lastMouseDown.parent.Edit
Fields[lastMouseDown.index].Font].Width);
```

```
            if lastMouseDown.parent.EditFields[lastMouseDown.index].Position < 0 then
lastMouseDown.parent.EditFields[lastMouseDown.index].Position := 0;
            if (length(lastMouseDown.parent.EditFields[lastMouseDown.index].Text)) <
lastMouseDown.parent.EditFields[lastMouseDown.index].Position then
              lastMouseDown.parent.EditFields[lastMouseDown.index].Position :=
length(lastMouseDown.parent.EditFields[lastMouseDown.index].Text)
          end else lastMouseDown.parent.EditFields[lastMouseDown.index].Position := 0;

          if (my <> lastMouseDown.parent.EditFields[lastMouseDown.index].y) then
          begin
            lastMouseDown.parent.EditFields[lastMouseDown.index].Line   := trunc((mY -
(lastMouseDown.parent.EditFields[lastMouseDown.index].Y+lastMouseDown.parent.Y +
lastMouseDown.parent.EditFields[lastMouseDown.index].plusx))/lastMouseDown.parent.EditFields[
lastMouseDown.index].space);
            if lastMouseDown.parent.EditFields[lastMouseDown.index].Line < 0 then
lastMouseDown.parent.EditFields[lastMouseDown.index].Line := 0;
            if (High(lastMouseDown.parent.EditFields[lastMouseDown.index].Stringlist)) <
lastMouseDown.parent.EditFields[lastMouseDown.index].line then
              lastMouseDown.parent.EditFields[lastMouseDown.index].line :=
High(lastMouseDown.parent.EditFields[lastMouseDown.index].Stringlist)
          end else lastMouseDown.parent.EditFields[lastMouseDown.index].line := 0;
          if lastMouseDown.Parent.EditFields[lastMouseDown.Index].position >
length(lastMouseDown.Parent.EditFields[lastMouseDown.Index].Stringlist[lastMouseDown.Parent.E
ditFields[lastMouseDown.Index].line]) then
            lastMouseDown.Parent.EditFields[lastMouseDown.Index].position :=
length(lastMouseDown.Parent.EditFields[lastMouseDown.Index].Stringlist[lastMouseDown.Parent.E
ditFields[lastMouseDown.Index].line]);
        end;
     15: begin
          lastMouseDown.Parent.Frames[lastMouseDown.Index].whichchecked := trunc((mX-
(lastMouseDown.X +
lastMouseDown.Parent.X))/lastMouseDown.Parent.Frames[lastMouseDown.Index].tabwidth);
          if lastMouseDown.Parent.Frames[lastMouseDown.Index].whichchecked >
High(lastMouseDown.Parent.Frames[lastMouseDown.Index].Frames) then
            lastMouseDown.Parent.Frames[lastMouseDown.Index].whichchecked :=
High(lastMouseDown.Parent.Frames[lastMouseDown.Index].Frames);
          for i := 0 to High(lastMouseDown.Parent.Frames[lastMouseDown.Index].Frames) do
            lastMouseDown.Parent.Frames[lastMouseDown.Index].Frames[i].Window.Visible :=
false;
          lastMouseDown.Parent.Frames[lastMouseDown.Index].Frames[lastMouseDown.Parent.Fra
mes[lastMouseDown.Index].whichchecked].Window.Visible := true;
        end;
    end;

    clickevent := true;
    if lastMouseDown.onMouseDownevent then
lastMouseDown.onMouseDown(mX,mY,lastMouseDown,mButton);
  end;

  if lastMouseDown <> nil then
  begin
    //set Z for all Windows -> reihenfolge der fenster.
```

```
    if GUIclass.active.Parent <> lastMouseDown.Parent then
      SetZforWindows;

    //set active
    active := lastMouseDown;
    if active.objecttype = 30 then
      active := active.Parent.PopupMenus[active.tag];
  end;
end;

procedure TGUIclass.MouseUp(mX,mY, mButton: integer);
var
  i,j,k: integer;
  oldlastMouseup: TGUIObject;
  tempfloat: single;
begin
  if dragEvent then    //die drags durchschauen
  begin
    dragevent := false;
    if lastMouseDown.afterdragevent then lastMouseDown.afterdrag(mX,mY,lastMouseDown, 1);
  end;

  if clickEvent then
  begin
    case lastMouseDown.objecttype of
      0:  lastMouseDown.Parent.Buttons[lastMouseDown.Index].Pressed := false;
      5:  lastMouseDown.Parent.Checkboxes[lastMouseDown.Index].Pressed := false;
      6:  lastMouseDown.Parent.RadioButtons[lastMouseDown.Index].Pressed := false;
      8:  lastMouseDown.Parent.MinimizeButton.pressed := false;
      9:  begin
            lastMouseDown.Parent.MaximizeButton.pressed := false;
            //lastMouseDown.Parent.MaximizeButton.normalized := not
lastMouseDown.Parent.MaximizeButton.normalized;
          end;
     10: lastMouseDown.Parent.CloseButton.pressed := false;
    end;
    if lastMouseDown.objecttype = 7 then
    begin
      if (lastMouseDown.X<=mX) and (lastMouseDown.Y<=mY) and
        ((lastMouseDown.X + lastMouseDown.Width)>=mX) and
        ((lastMouseDown.Y + lastMouseDown.Width)>=mY) then
       if lastMouseDown.onclickevent then
         lastMouseDown.onclick(mX,mY, lastMouseDown,mButton)
    end
    else
    if lastMouseDown.objecttype = 30 then
    begin
       if lastMouseDown.Parent.PopupMenus[lastMouseDown.X].MouseonElement(mX,mY) then
         if
lastMouseDown.Parent.PopupMenus[lastMouseDown.X].PopUpMenuItems[lastMouseDown.Parent.P
opupMenus[lastMouseDown.X].whichchecked] = lastMousedown then    //x wird "missbraucht"
          if lastMouseDown.onclickevent then
```

```
          lastMouseDown.onclick(mX,mY, lastMouseDown,mButton);
    end else
      if lastMouseDown.MouseonElement(mx,my) then
      begin
        case lastMouseDown.objecttype of
          5:  if (mbutton = 1) then
lastMouseDown.parent.Checkboxes[lastMouseDown.index].Checked := not
lastMouseDown.parent.Checkboxes[lastMouseDown.Index].Checked;
          6:  if (mbutton = 1) then
              begin
                for k := 1 to
High(lastMouseDown.parent.RadioButtons[lastMouseDown.index].ParentGroup.RadioButtons) do
                  lastMouseDown.parent.RadioButtons[lastMouseDown.index].ParentGroup.RadioButton
s[k].Checked := false;
                lastMouseDown.parent.RadioButtons[lastMouseDown.parent.RadioButtonGroups[lastMo
useDown.parent.RadioButtons[lastMouseDown.index].group].whichchecked].Checked := false;
                lastMouseDown.parent.RadioButtons[lastMouseDown.index].Checked := true;//not
lastMouseDown.parent.RadioButtons[lastMouseDown.index].Checked;
                lastMouseDown.parent.RadioButtonGroups[lastMouseDown.parent.RadioButtons[lastMo
useDown.index].group].whichchecked :=
lastMouseDown.parent.RadioButtons[lastMouseDown.index].GroupIndex;
              end;
        end;
        if lastMouseDown.onclickevent then
        begin
          if lastMouseDown.objecttype = 9 then lastMouseDown.Parent.MaximizeButton.maximized
:= not lastMouseDown.Parent.MaximizeButton.maximized;
          lastMouseDown.onclick(mX,mY, lastMouseDown,mButton);
        end;
      end;
    clickEvent := false;
  end;

  oldlastMouseUp := lastMouseUp;
  lastMouseUp := nil;
  for j:=0 to High(Windows) do
  begin
    if Windows[j].MouseonElement(mX, mY) then
    begin
      if lastmouseUp = nil then
        lastmouseUp  := Windows[j]
      else
        if lastmouseUp.Z < Windows[j].Z then
          lastmouseUp  := Windows[j];


      // ***die einzelnen Elemente berechnen***
      //panels
      for i :=0 to High(Windows[j].Panels) do
        if Windows[j].Panels[i].MouseonElement(mX, mY) then
          if lastMouseUp.Z < Windows[j].Panels[i].Z then
            lastMouseUp := Windows[j].Panels[i];
```

```
//ProgressBars
for i :=0 to High(Windows[j].ProgressBars) do
  if Windows[j].ProgressBars[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].ProgressBars[i].Z then
      lastMouseUp := Windows[j].ProgressBars[i];

//Edits
for i :=0 to High(Windows[j].Edits) do
  if Windows[j].Edits[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].Edits[i].Z then
    begin
      lastMouseUp := Windows[j].Edits[i];
    end;

//Buttons
for i :=0 to High(Windows[j].Buttons) do
  if Windows[j].Buttons[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].Buttons[i].Z then
    begin
      lastMouseUp := Windows[j].Buttons[i];
    end;

//Text
for i :=0 to High(Windows[j].Text) do
  if Windows[j].Text[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].text[i].Z then
      lastMouseUp := Windows[j].text[i];

//Checkboxes
for i :=0 to High(Windows[j].Checkboxes) do
  if Windows[j].Checkboxes[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].Checkboxes[i].Z then
      lastMouseUp := Windows[j].Checkboxes[i];

//RadioButtons
for i:=0 to High(Windows[j].RadioButtons) do
  if Windows[j].RadioButtons[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].RadioButtons[i].Z then
      lastMouseUp := Windows[j].RadioButtons[i];

//Frames
for i := 0 to High(Windows[j].Frames) do
  if Windows[j].Frames[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].Frames[i].Z then
      lastMouseUp := Windows[j].Frames[i];

//Images
for i := 0 to High(Windows[j].Images) do
  if Windows[j].Images[i].MouseonElement(mX, mY) then
    if lastMouseUp.Z < Windows[j].Images[i].Z then
      lastMouseUp := Windows[j].Images[i];
```

```
    {//Comboboxes
    for i := 0 to High(Windows[j].Comboboxes) do
      if Windows[j].Comboboxes[i].MouseonElement(mX, mY) then
        if lastMouseUp.Z < Windows[j].Comboboxes[i].Z then
          lastMouseUp := Windows[j].Comboboxes[i];       }

    //EditFields
    for i := 0 to High(Windows[j].EditFields) do
      if Windows[j].EditFields[i].MouseonElement(mX, mY) then
        if lastMouseUp.Z < Windows[j].EditFields[i].Z then
          lastMouseUp := Windows[j].EditFields[i];

    //TextFields
    for i := 0 to High(Windows[j].TextFields) do
      if Windows[j].TextFields[i].MouseonElement(mX, mY) then
        if lastMouseUp.Z < Windows[j].TextFields[i].Z then
          lastMouseUp := Windows[j].TextFields[i];
  end;

  //Comboboxes
  if Windows[j].Visible then
    for i :=0 to High(Windows[j].Comboboxes) do
        if Windows[j].Comboboxes[i].MouseonElement(mX, mY) then
          if lastMouseUp = nil then
          begin
            if Windows[j].Comboboxes[i].opened then
            begin
              lastMouseUp  :=
Windows[j].Comboboxes[i].ComboboxItems[Windows[j].Comboboxes[i].mouseon];
              if 0 < ((mY - (Windows[j].y + Windows[j].Comboboxes[i].Y +
Windows[j].Comboboxes[i].Height))/Windows[j].Comboboxes[i].Space) then
              begin
                Windows[j].Comboboxes[i].whichchecked := Windows[j].Comboboxes[i].mouseon;
                Windows[j].Comboboxes[i].opened := false;
              end;
            end else
            begin
              lastMouseDown  := Windows[j].Comboboxes[i];
            end;
          end
          else
            if lastMouseUp.Z < Windows[j].Comboboxes[i].Z then
            begin
              if Windows[j].Comboboxes[i].opened then
              begin
                lastMouseUp  :=
Windows[j].Comboboxes[i].ComboboxItems[Windows[j].Comboboxes[i].mouseon];
                if 0 < ((mY - (Windows[j].y + Windows[j].Comboboxes[i].Y +
Windows[j].Comboboxes[i].Height))/Windows[j].Comboboxes[i].Space) then
                begin
                  Windows[j].Comboboxes[i].whichchecked := Windows[j].Comboboxes[i].mouseon;
```

```
              Windows[j].Comboboxes[i].opened := false;
            end;
          end else
          begin
            lastMouseDown  := Windows[j].Comboboxes[i];
          end;
        end;

  //PopUpMenus
  if Windows[j].Visible then
    for i := 0 to High(Windows[j].PopUpMenus) do
      if Windows[j].PopUpMenus[i].MouseonElement(mX, mY) then
        if lastMouseUp = nil then
        begin
          lastMouseUp :=
Windows[j].PopupMenus[i].PopUpMenuItems[Windows[j].PopupMenus[i].whichchecked];
          Windows[j].PopUpMenus[i].opened := false;
        end else
          if lastMouseUp.Z < Windows[j].PopUpMenus[i].Z then
          begin
            lastMouseUp :=
Windows[j].PopupMenus[i].PopUpMenuItems[Windows[j].PopupMenus[i].whichchecked];
            Windows[j].PopUpMenus[i].opened := false;
          end;
 end;

 if lastMouseUP <> nil then
 begin
   if lastMouseUp.onMouseupevent then lastMouseUp.onMouseup(mX,mY, lastMouseUp,mButton);
   if (lastMouseDown = lastMouseUp) and (mbutton=1) and (lastMouseUp.objecttype <> 30) and
      (lastMouseUp.objecttype <> 31) then
   begin
     oldlastMouseUp.active := false;
     lastMouseUp.active := true;
   end else
     lastMouseUp := oldLastmouseUp;
 end
 else lastMouseUp := oldLastmouseUp;
end;

procedure TGUIclass.KeyDown(Key: Word);   //neu: word wegen SDL
var
 s: string;
 edittext: string;
begin
 if lastMouseUp.onkeydownevent then
   lastMouseUp.onkeydown(Key, lastMouseUp);
 lastkeydown := key;
 if lastMouseUp.objecttype = 4 then  //edit zeugs...
 begin
   showcursor := true;
   edittext := lastMouseUp.Parent.Edits[lastMouseUp.Index].text;
```

```
   s := GetKey(Key);
  {$IFDEF sdl}
   case Key of
    //   8  :    Key :=    8;  //backspace
    //  13  :    Key :=   13;  //enter
     303  :    Key :=   16;  //shift     (right)
     304  :    Key :=   16;  //shift     (left)
     307  :    Key :=   18;  //alt      (right)
     308  :    Key :=   18;  //alt      (left)
     301  :    Key :=   20;  //capslock
    //  27  :    Key :=   27;  //ESC
     276  :    Key :=   37;  //LEFT
     273  :    Key :=   38;  //UP
     275  :    Key :=   39;  //RIGHT
     274  :    Key :=   40;  //DOWN
     127  :    Key :=   46;  //DEL
   end;
  {$ENDIF}
   if s = '' then
    case Key of
      8:  if lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition <> 0 then  //backspace
          begin
            lastMouseUp.Parent.edits[lastMouseUp.Index].Text := copy(edittext, 0,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition-1)
                                                +copy(edittext,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+1
                                                , length(edittext) -
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+1);
            dec(lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition);
          end;
      //13:  (enter)
      27: lastMouseUp := lastMouseUp.Parent;  //ESC
      37: if lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition > 0 then
dec(lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition); //left
      39: if length(lastMouseUp.Parent.Edits[lastMouseUp.Index].Text) >
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition then
inc(lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition); //right
      46: if length(edittext) <> lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition then
//DEL
          begin
            lastMouseUp.Parent.edits[lastMouseUp.Index].Text := copy(edittext, 0,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition)
                                                +copy(edittext,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+2
                                                , length(edittext) -
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+1);
          end else
          begin
            if length(edittext) > 0 then
            begin
              lastMouseUp.Parent.edits[lastMouseUp.Index].Text := copy(edittext, 0,
length(edittext) - 1);
```

```
            dec(lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition);
          end;
        end;
    end
    else
    begin
      lastMouseUp.Parent.Edits[lastMouseUp.Index].Text := copy(edittext, 0,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition)
                                                + s +
                                                copy(edittext,
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+1, length(edittext) -
lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition+1);
      inc(lastMouseUp.Parent.Edits[lastMouseUp.Index].MousePosition);
    end;
  end;
  if lastMouseUp.objecttype = 11 then  //EditField zeugs...
  begin
    self.showcursor := true;
    edittext :=
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields[lastMo
useUp.Index].line];
    s := GetKey(Key);
  {$IFDEF sdl}
    case Key of
     //   8  :    Key :=    8;  //backspace
     //  13  :    Key :=   13;  //enter
      303  :    Key :=   16;  //shift     (right)
      304  :    Key :=   16;  //shift     (left)
      307  :    Key :=   18;  //alt      (right)
      308  :    Key :=   18;  //alt      (left)
      301  :    Key :=   20;  //capslock
     //  27  :    Key :=   27;  //ESC
      276  :    Key :=   37;  //LEFT
      273  :    Key :=   38;  //UP
      275  :    Key :=   39;  //RIGHT
      274  :    Key :=   40;  //DOWN
      127  :    Key :=   46;  //DEL
    end;
  {$ENDIF}
    if s = '' then
      case Key of
        8:  if not lastMouseUp.Parent.EditFields[lastMouseUp.Index].readonly then
            if lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position <> 0 then
            begin
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFiel
ds[lastMouseUp.Index].line] := copy(edittext, 0,
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position-1)
                                                    +copy(edittext,
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+1, length(edittext) -
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+1);
              dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position);
            end else
```

```
          begin
            if lastMouseUp.Parent.EditFields[lastMouseUp.Index].line <> 0 then
            begin
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].position :=
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line-1]);
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFi
elds[lastMouseUp.Index].line - 1] :=
               lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.Edit
Fields[lastMouseUp.Index].line - 1] +
               lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.Edit
Fields[lastMouseUp.Index].line];
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].DeleteLine(lastMouseUp.Parent.Edit
Fields[lastMouseUp.Index].line);
              dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
            end;
          end;
       13: if not lastMouseUp.Parent.EditFields[lastMouseUp.Index].readonly then
//(enter)
          begin
            if lastMouseUp.Parent.EditFields[lastMouseUp.Index].position <
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]) then
            begin
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].MakeNewLine(lastMouseUp.Parent.E
ditFields[lastMouseUp.Index].line + 1);
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFi
elds[lastMouseUp.Index].line + 1] :=
              copy(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.
EditFields[lastMouseUp.Index].line],lastMouseUp.Parent.EditFields[lastMouseUp.Index].position +
1
              , length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist) -
lastMouseUp.Parent.EditFields[lastMouseUp.Index].position + 2);
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFi
elds[lastMouseUp.Index].line] :=
              copy(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.
EditFields[lastMouseUp.Index].line],0,
lastMouseUp.Parent.EditFields[lastMouseUp.Index].position);
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].position := 0;
              inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
            end else
            begin
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].MakeNewLine(lastMouseUp.Parent.E
ditFields[lastMouseUp.Index].line + 1);
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].position := 0;
              inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
            end;
          end;
       27: lastMouseUp := lastMouseUp.Parent;  //ESC
       37: if lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position > 0 then
            dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position)
          else
```

```
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].Line > 0 then
          begin
            dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Line);
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position :=
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].Line]);
          end;
      38: begin                                                    //UP
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].line > 0 then
            dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].position >
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]) then
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].position :=
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]);
        end;
      39: if lastMouseUp.Parent.EditFields[lastMouseUp.Index].position <
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]) then
          inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position)
        else
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].Line <
High(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist) then
          begin
            inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Line);
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position := 0;
          end;
      40: begin                                                  //DOWN
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].line <
high(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist) then
            inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
          if lastMouseUp.Parent.EditFields[lastMouseUp.Index].position >
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]) then
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].position :=
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]);
          end;
      46: if not lastMouseUp.Parent.EditFields[lastMouseUp.Index].readonly then
          if length(edittext) <> lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position then
          begin
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFiel
ds[lastMouseUp.Index].line] :=
                                                  copy(edittext, 0,
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position)
                                                  +copy(edittext,
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+2, length(edittext) -
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+1);
          end else
          begin
            if High(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist) =
```

```
lastMouseUp.Parent.EditFields[lastMouseUp.Index].line then
          begin
            if length(edittext) <> 0 then
            begin
              lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.Edit
Fields[lastMouseUp.Index].line] := copy(edittext, 0, length(edittext) - 1);
              dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position);
            end else
            begin
              if lastMouseUp.Parent.EditFields[lastMouseUp.Index].line <> 0 then
              begin
                lastMouseUp.Parent.EditFields[lastMouseUp.Index].DeleteLine(lastMouseUp.Parent.E
ditFields[lastMouseUp.Index].line);
                dec(lastMouseUp.Parent.EditFields[lastMouseUp.Index].line);
                lastMouseUp.Parent.EditFields[lastMouseUp.Index].position :=
length(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields
[lastMouseUp.Index].line]);
              end;
            end;
          end else
          begin
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFi
elds[lastMouseUp.Index].line]
            :=
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields[lastMo
useUp.Index].line] +
lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields[lastMo
useUp.Index].line+1];
            lastMouseUp.Parent.EditFields[lastMouseUp.Index].DeleteLine(lastMouseUp.Parent.Edit
Fields[lastMouseUp.Index].line + 1);
          end;
        end;
    end
    else
      if not lastMouseUp.Parent.EditFields[lastMouseUp.Index].readonly then
      begin
        lastMouseUp.Parent.EditFields[lastMouseUp.Index].Stringlist[lastMouseUp.Parent.EditFields[
lastMouseUp.Index].line] :=
          copy(edittext, 0, lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position)
          + s +
          copy(edittext, lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+1,
length(edittext) - lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position+1);
        inc(lastMouseUp.Parent.EditFields[lastMouseUp.Index].Position);
      end;
 end;

 case Key of
   16:  Shift := true;
   18:  Alt   := true;
   20:  CapsLock := not CapsLock;
 end;
 keyisdown := true;
```

```
end;

procedure TGUIclass.KeyUp(Key: Word);
var
  i: integer;
begin
  {$IFDEF sdl}
    case Key of
      303  :    Key :=   16;  //shift      (right)
      304  :    Key :=   16;  //shift      (left)
      307  :    Key :=   18;  //alt       (right)
      308  :    Key :=   18;  //alt       (left)
      301  :    Key :=   20;  //capslock
      276  :    Key :=   37;  //LEFT
      273  :    Key :=   38;  //UP
      275  :    Key :=   39;  //RIGHT
      274  :    Key :=   40;  //DOWN
      127  :    Key :=   46;  //DEL
    end;
  {$ENDIF}
  if lastMouseUp.onkeyUpevent then
    lastMouseUp.onKeyUp(Key, lastMouseUp);

  case Key of
    16:  Shift := false;
    18:  Alt   := false;
  end;
end;

function  TGUIclass.LoadFont(index: integer; fontname: string): boolean;
var
  i : GLuint;
  cx, cy, xs : GLfloat;
  W : Byte;
  fs : TMemoryStream;
  fdl: gluint;
  FontWidth : Array[0..255] of Byte;

begin
  result := true;
  LoadTexture(fontname + '.tga', fonts[index].graphic, false);    //fontname sollte OHNE .tga
angegeben werden.

  fs:=TMemoryStream.Create;   // Load the character widths from the font.fnt file
  fs.LoadFromFile(fontname+'.fnt');
  fs.Position:=0;
  for i :=0 to 255 do
  begin
    fs.ReadBuffer(W,1);
    FontWidth[i] :=W;
  end;
  fs.free;
```

```
  glBindTexture(GL_TEXTURE_2D, fonts[index].graphic);
  glPushMatrix;
  // create the font display list
  fonts[index].list := glGenLists(256);             // Storage For 128 Characters
  for i := 0 to 255 do
  begin
    cx := (i mod 16) / 16;                          // X Position Of Current Character
    cy := (i div 16) / 16;                          // Y Position Of Current Character
    glNewList(fonts[index].list+i, GL_COMPILE);      // Start Building A List
    glBegin(GL_QUADS);
      xs := (16-FontWidth[i])/2/256;
      glTexCoord2f(cx+xs,1-cy-0.0625);                        glVertex2i(0, 16);
      glTexCoord2f(cx+xs + FontWidth[i]/256 + 1/512, 1-cy-0.0625);  glVertex2i( FontWidth[i],
16);
      glTexCoord2f(cx+xs + FontWidth[i]/256 + 1/512, 1-cy);        glVertex2i( FontWidth[i], 0);
      glTexCoord2f(cx+xs,1-cy);                                 glVertex2i(0, 0);
    glEnd();
    glTranslatef(FontWidth[i]+1, 0, 0);
    glEndList();
  end;
  glPopMatrix;
end;

procedure TGuiclass.LoadFont2(index: integer; path: string; scale: single);
type
  TFontRect = record
    x1,y1,x2,y2: single;
  end;
  TFontCoords = array[0..255] of TFontRect;
var
  i: integer;
  FontCoord:  TFontCoords;
  Stream:     TFileStream;
begin
  //Stream auslesen...
  Stream := TFileStream.Create((path + '.fci'), fmOpenRead);
  Stream.Read(FontCoord, sizeof(FontCoord));
  Stream.Free;

  //Bild einlesen...
  LoadTexture(path + '.tga', fonts[index].graphic, false); //fontname sollte OHNE .bmp angegeben
werden.

  glPushMatrix;
  //Platz machen für 256 Listen im Speicher...
  fonts[Index].list := glGenLists(256);

  fonts[Index].width  := round((FontCoord[88].x2 - FontCoord[88].x1)*512*scale);
  fonts[Index].height := round((FontCoord[88].y2 - FontCoord[88].y1)*512*scale);
  for i := 0 to 255 do
  begin
```

```
   glNewList(fonts[Index].list+i, GL_COMPILE);       // Start Building A List
     glBegin(GL_QUADS);
       glTexCoord2f(FontCoord[i].x1,1-FontCoord[i].y1);  glVertex2i(  0,                0
);
       glTexCoord2f(FontCoord[i].x1,1-FontCoord[i].y2);  glVertex2i(  0,
fonts[Index].height   );
       glTexCoord2f(FontCoord[i].x2,1-FontCoord[i].y2);  glVertex2i(  fonts[Index].width,
fonts[Index].height   );
       glTexCoord2f(FontCoord[i].x2,1-FontCoord[i].y1);  glVertex2i(  fonts[Index].width,  0
);
     glEnd();
     glTranslatef(fonts[Index].width, 0, 0);
   glEndList();
  end;
  glPopMatrix;
end;

function  TGUIclass.AddWindow(wx,wy, wwidth,wheight, wskin: integer; wcaption : string;
wgraphic : string = 'data\skins\default.tga'): integer;
var
  i,j: integer;
  free: boolean;
begin
  free:=false;
  for i := 0 to High(windows) do
    if not windows[i].used then
    begin
      free:=true;
      j:=i;
    end;
  if not free then
  begin
    setlength(windows, High(windows)+2);
    j:=High(windows);
    windows[j] := TWindow.Create;
  end;

  setlength(Winorder, length(Windows));
  Winorder[j] := -2;

  result := j;
  with windows[j] do
  begin
    PSelf := windows[j];
    Parent := windows[j];
    setlength(Buttons,       1);
    setlength(Text,          1);
    setlength(RadioButtons,  1);
    setlength(RadioButtonGroups, 1);
    setlength(Checkboxes,    1);
    setlength(Edits,         1);
    setlength(Panels,        1);
```

```
setlength(ProgressBars,   1);
setlength(Images,         1);
setlength(Frames,         1);
setlength(PopUpMenus,     1);
setlength(Comboboxes,     1);
setlength(EditFields,     1);
setlength(TextFields,     1);

Buttons[0]         := TButton.Create;
Text[0]            := TText.Create;
RadioButtons[0]    := TRadioButton.Create;
Checkboxes[0]      := TCheckbox.Create;
Edits[0]           := TEdit.Create;
Panels[0]          := TPanel.Create;
ProgressBars[0]    := TProgressBar.Create;
Images[0]          := TImage.Create;
Frames[0]          := TFrames.Create;
EditFields[0]      := TEditField.Create;
TextFields[0]      := TTextField.Create;

RadioButtonGroups[0] := TRadioButtonGroup.Create;
setlength(RadioButtonGroups[0].RadioButtons, 1);

PopUpMenus[0]      := TPopUpMenu.Create;
setlength(PopUpMenus[0].PopUpMenuItems, 1);              //sollte raus
PopUpMenus[0].PopupMenuItems[0] := TPopUpMenuItem.Create;  //sollte raus

Comboboxes[0]      := TCombobox.Create;
setlength(Comboboxes[0].Comboboxitems, 1);               //sollte raus
Comboboxes[0].ComboboxItems[0] := TComboboxItem.Create;    //sollte raus

MinimizeButton     := TMinimizeButton.Create;
MaximizeButton     := TMaximizeButton.Create;
CloseButton        := TCloseButton.Create;

Index := j;
used := true;
Objecttype := 7;
X :=wX;
Y :=wY;
_Z :=0;
Width  :=wWidth;
Height :=wHeight;
_Visible := false;
Captionbar := false;
Child := false;
Background := true;
Color := cWhite;
Skin:= wSkin; //default skin
caption:= wcaption;
LoadTexture(wgraphic, graphic, false);
graphicpath := wgraphic;
```

```
    CaptionBarHeight := 26; //default captionbarheight
    isTab := false;
    Fontcolor := cWhite;

    setlength(ChildWindows, 1);
    ChildWindows[0].Index := 0;
    ChildWindows[0].used := false;

    ChangeCaptionBarButtons;
    MinimizeButton._Visible := false;
    MaximizeButton._Visible := false;
    CloseButton._Visible := false;

    MaximizeButton.maximized := false;
    MinimizeButton.minimized  := false;

    MinimizeButton.Z := 1.45;
    MaximizeButton.Z := 1.45;
    CloseButton.Z := 1.45;

    MinimizeButton.used := true;
    MaximizeButton.used := true;
    CloseButton.used := true;
  end;
  setZforWindows;
  Windows[j].ChangeCaptionBarButtons;
end;

function  TGUIclass.AddText(tx,ty,tz, tFont : integer; ttext: string; tColor: TColor): integer;
var
  i,j: integer;
  free: boolean;
begin
  free:=false;
  for i := 0 to High(Text) do
    if not Text[i].used then
    begin
      free:=true;
      j:=i;
    end;
  if not free then
  begin
    setlength(Text, High(Text)+2);
    j:=High(Text);
    GUIclass.Text[j] :=  GUI.TText.Create;
  end;

  Text[j].objecttype := 50;
  Text[j].X := tx;
  Text[j].Y := ty;
  Text[j].Z := tz;
  Text[j].Text := ttext;
```

```
  Text[j].Color := tColor;
  //Text[j].Update(ttext);
  Text[j].Visible := true;
  Text[j].used := true;
  Text[j].font := tFont;

  result := j;
end;

function  TGUIclass.LoadSkinTexture(path: string; var Texture: cardinal): integer; //gives filetype
back...
begin
  if FileExists(path + '.bmp') then
  begin
    LoadTexture(path + '.bmp', Texture, false);     //filetype 1
    result := 1;
  end else
    if FileExists(path + '.tga') then
    begin
      LoadTexture(path + '.tga', Texture, false);   //filetype 2
      result := 2;
    end else
      if FileExists(path + '.jpg') then
      begin
        LoadTexture(path + '.jpg', Texture, false); //filetype 3
        result := 3;
      end else
        if FileExists(path + '.jpeg') then
        begin
          LoadTexture(path + '.jpeg', Texture, false); //filetype 3
          result := 3;
        end else
          begin
            {$ifdef showerrors}
            showmessage('Format is unknown or it does not exist!');      //filetype none (-1)
            {$endif}
            result := - 1;
          end;
end;

function  TGUIclass.AddSkin(pathdir: string): integer;
var
  i,j: integer;
  free: boolean;
begin
  free:=false;
  for i := 0 to High(Skins) do
    if not Skins[i].used then
    begin
      free:=true;
      j:=i;
    end;
```

```
  if not free then
  begin
    setlength(Skins, High(Skins)+2);
    j:=High(Skins);
  end;

  result := j;
  Skins[j].used := true;

  with Skins[j] do
  begin
    LoadSkinTexture(pathdir + 'panel', panel);
    LoadSkinTexture(pathdir + 'edit', edit);
    LoadSkinTexture(pathdir + 'captionbar', captionbar);
    LoadSkinTexture(pathdir + 'progress_ground', progress_ground);
    LoadSkinTexture(pathdir + 'progressbar', progressbar);
    LoadSkinTexture(pathdir + 'ground', ground);
    LoadSkinTexture(pathdir + 'editfield', editfield);

    LoadSkinTexture(pathdir + 'button', button);
    LoadSkinTexture(pathdir + 'button_c', button_c);
    LoadSkinTexture(pathdir + 'button_m', button_m);

    LoadSkinTexture(pathdir + 'frame_c', frame_c);
    LoadSkinTexture(pathdir + 'frame', frame);

    LoadSkinTexture(pathdir + 'check', check);
    LoadSkinTexture(pathdir + 'check_m', check_m);
    LoadSkinTexture(pathdir + 'check_c', check_c);
    LoadSkinTexture(pathdir + 'check_m_c', check_m_c);

    LoadSkinTexture(pathdir + 'radio', radio);
    LoadSkinTexture(pathdir + 'radio_m', radio_m);
    LoadSkinTexture(pathdir + 'radio_c', radio_c);
    LoadSkinTexture(pathdir + 'radio_m_c', radio_m_c);

    LoadSkinTexture(pathdir + 'minimize', minimize);
    LoadSkinTexture(pathdir + 'minimize_m', minimize_m);
    LoadSkinTexture(pathdir + 'normalize', normalize);
    LoadSkinTexture(pathdir + 'normalize_m', normalize_m);
    LoadSkinTexture(pathdir + 'maximize', maximize);
    LoadSkinTexture(pathdir + 'maximize_m', maximize_m);
    LoadSkinTexture(pathdir + 'close', close);
    LoadSkinTexture(pathdir + 'close_m', close_m);
    LoadSkinTexture(pathdir + 'default', default);

    LoadSkinTexture(pathdir + 'combobox_button', Combobox_button);
    LoadSkinTexture(pathdir + 'combobox', Combobox);
    LoadSkinTexture(pathdir + 'combo_ground', combo_ground);
  end;
end;
```

```
function  TGUIclass.AddFont(path: string; scale: single = 1): integer;
var
  i,j: integer;
  free: boolean;
begin
  free:=false;
  for i := 0 to High(Fonts) do
    if not Fonts[i].used then
    begin
      free:=true;
      j:=i;
    end;
  if not free then
  begin
    setlength(Fonts, High(Fonts)+2);
    j:=High(Fonts);
  end;

  if copy(path,length(path) - 2,3) = 'fci' then
    LoadFont2(j,copy(path,0,length(path)-4), scale)
  else
  begin
    if not LoadFont(j, copy(path,0,length(path)-4)) then
    begin
    {$ifdef showerrors}
      showmessage('Konnte die Schrift: "' + path + '" nicht laden');
    {$endif}
    end;
    Fonts[j].Height := strtoint(copy(path, length(path)-5, 2));
    Fonts[j].width := round(Fonts[j].Height*0.7);
  end;
  Fonts[j].used := true;
  Fonts[j].name := copy(path, 0, length(path)-2);
  result := j ;
end;

//Delete Proceduren
procedure TGUIclass.DeleteWindow(index: integer);
var
  i : integer;
begin
  with Windows[index] do
  begin
    visible := false;
    used := false;
    //FreeComponents;
  end;
end;

procedure TGUIclass.DeleteText(index: integer);
begin
  Text[index].visible := false;
```

```
  Text[index].used := false;
end;

procedure TGUIclass.DeleteSkin(index: integer);
begin
  Skins[index].used := false;
end;

procedure TGUIclass.DeleteFont(index: integer);
begin
  Fonts[index].used := false;
  glDeleteLists(Fonts[index].list, 256);
end;

{------------------------------------------------------------------------------}
{------------------------ Load & Save -----------------------------------------}
{------------------------------------------------------------------------------}
procedure TGUIObject.LoadFromStream(S: TBinaryFile);
var
  p: integer;
begin
  S.Read(_X);
  S.Read(_Y);
  S.Read(_Z);
  S.Read(_Width);
  S.Read(_Height);
  S.Read(Skin);
  S.Read(Font);
  S.Read(dragevent);
  S.Read(Used);
  S.Read(_Visible);
  S.Read(p);
  if p > (-1) then
  begin
    PopUpMenu := Parent.PopUpMenus[p];
    hasPopUpMenu := true;
  end;
end;

procedure TGUIObject.SaveToStream(S: TBinaryFile);
var
  p: integer;
begin
  S.Write(_X);
  S.Write(_Y);
  S.Write(_Z);
  S.Write(_Width);
  S.Write(_Height);
  S.Write(Skin);
  S.Write(Font);
  S.Write(dragevent);
  S.Write(Used);
```

```
  S.Write(_Visible);
  if hasPopUpMenu then
    p := PopUpMenu.Index
  else
    p := -1;
  S.Write(p);
end;

procedure TWindow.LoadFromStream(S: TBinaryFile);
var
  //r: TSaveRecord;
  _high: word;
  i: integer;
  p: integer;
begin
  S.Read(_high);
  for i := 0 to _high do
  begin
    AddPopUpMenu(0,0,'');
    PopupMenus[i].LoadFromStream(S);
    PopupMenus[i].LoadFromStream_Individual(S);
  end;

  {S.S.Read(r,SizeOf(TSaveRecord));

  objecttype := r.objecttype;
  X := r.X;
  Y := r.Y;
  Z := r.Z;
  Used := r.used;
  Visible := r.visible;
  Width := r.Width;
  Height := r.Height;
  if r.PopUpMenu > (-1) then
  begin
    PopUpMenu := Parent.PopUpMenus[r.PopUpMenu];
    hasPopUpMenu := true;
  end;

  skin := r.skin;
  font := r.font;}

  S.Read(_X);
  S.Read(_Y);
  S.Read(_Z);
  S.Read(_Width);
  S.Read(_Height);
  S.Read(Skin);
  S.Read(Font);
  S.Read(dragevent);
  S.Read(Used);
  S.Read(_Visible);
```

```
  S.Read(p);
  if p > (-1) then
  begin
    PopUpMenu := PopUpMenus[p];
    hasPopUpMenu := true;
  end;

  LoadFromStream_Individual(S);
end;

procedure TWindow.SaveToStream(S: TBinaryFile);
var
  //r: TSaveRecord;
  p: integer;
  _high: word;
  i: integer;
begin
  _high := High(PopupMenus);
  S.Write(_high);
  for i := 0 to High(PopupMenus) do
  begin
    PopupMenus[i].SaveToStream(S);
    PopupMenus[i].SaveToStream_Individual(S);
  end;

  {r.objecttype := objecttype;
  r.X := X;
  r.Y := Y;
  r.Z := Z;
  r.Width := Width;
  r.Height := Height;
  if hasPopUpMenu then
    r.PopUpMenu := PopUpMenu.Index
  else
    r.PopUpMenu := -1;
  r.skin := skin;
  r.font := font;
  r.visible := visible;
  r.used := used;
  S.S.Write(r,SizeOf(TSaveRecord));   }

  S.Write(_X);
  S.Write(_Y);
  S.Write(_Z);
  S.Write(_Width);
  S.Write(_Height);
  S.Write(Skin);
  S.Write(Font);
  S.Write(dragevent);
  S.Write(Used);
  S.Write(_Visible);
  if hasPopUpMenu then
```

```
    p := PopUpMenu.Index
  else
    p := -1;
  S.Write(p);

  SaveToStream_Individual(S);
end;

procedure TGUIObject.LoadFromStream_Individual(S: TBinaryFile);
begin
//"Pseudo" Routine, die von den meisten Komponenten überladen wird...
end;
procedure TGUIObject.SaveToStream_Individual(S: TBinaryFile);
begin
//"Pseudo" Routine, die von den meisten Komponenten überladen wird...
end;

procedure TImage.LoadFromStream_Individual(S: TBinaryFile);
begin
  S.Read(blending);
  S.Read(sfactor);
  S.Read(dfactor);
  S.S.Read(Color, SizeOf(TColor));
  S.Read(graphicpath);
  LoadTexture(graphicpath,graphic,false);
end;
procedure TImage.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(blending);
  S.Write(sfactor);
  S.Write(dfactor);
  S.S.Write(Color, SizeOf(TColor));
  S.Write(graphicpath);
end;
procedure TButton.LoadFromStream_Individual(S: TBinaryFile);
begin
  S.Read(plusx);
  S.Read(plusy);
  S.Read(Caption);
  S.Read(Pressed);
  S.S.Read(TextColor, SizeOf(TColor));
end;
procedure TButton.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(plusx);
  S.Write(plusy);
  S.Write(Caption);
  S.Write(Pressed);
  S.S.Write(TextColor, SizeOf(TColor));
end;
procedure TCheckbox.LoadFromStream_Individual(S: TBinaryFile);
begin
```

```
  S.Read(Checked);
  S.Read(Pressed);
end;
procedure TCheckbox.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Checked);
  S.Write(Pressed);
end;
procedure TRadioButton.LoadFromStream_Individual(S: TBinaryFile);
begin
  S.Read(Checked);
  S.Read(pressed);
  S.Read(Group);
  S.Read(GroupIndex);
  if GroupIndex > 0 then
  begin
    ParentGroup := Parent.RadioButtonGroups[Group];
    if High(ParentGroup.RadioButtons) < GroupIndex then
      setlength(ParentGroup.RadioButtons, GroupIndex + 1);
    ParentGroup.RadioButtons[GroupIndex] := Parent.RadioButtons[Index];
  end;
end;
procedure TRadioButton.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Checked);
  S.Write(pressed);
  S.Write(Group);
  S.Write(GroupIndex);
end;
procedure TTextField.LoadFromStream_Individual(S: TBinaryFile);
var
  str: string;
begin
  S.Read(Space);
  S.S.Read(TextColor, SizeOf(TColor));
  S.Read(str);
  Text := str
end;
procedure TTextField.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Space);
  S.S.Write(TextColor, SizeOf(TColor));
  S.Write(Text);
end;
procedure TEditField.LoadFromStream_Individual(S: TBinaryFile);
var
  str: string;
begin
  S.Read(Space);
  S.S.Read(TextColor, SizeOf(TColor));
  S.Read(readonly);
  S.Read(plusx);
```

```
  S.Read(plusy);
  S.Read(str);
  Text := str;
  line := 0;
  position := 0;
end;
procedure TEditField.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Space);
  S.S.Write(TextColor, SizeOf(TColor));
  S.Write(readonly);
  S.Write(plusx);
  S.Write(plusy);
  S.Write(Text);
end;
procedure TFrames.LoadFromStream_Individual(S: TBinaryFile);
var
  i,j: integer;
  _high: word;
begin
  S.Read(CaptionBarHeight);
  S.Read(tabwidth);
  S.Read(plusx);
  S.Read(plusy);
  S.S.Read(TextColor, SizeOf(TColor));
  whichchecked := 0;
  S.Read(_high);
  Frames[0].Window.LoadFromStream(S);
  for i := 1 to _high do
  begin
    j := GUIclass.AddWindow(0,0,0,0,0,'');
    AddFrame(Guiclass.Windows[j]);
    Frames[i].Window.LoadFromStream(S);
  end;
end;
procedure TFrames.SaveToStream_Individual(S: TBinaryFile);
var
  i: integer;
  _high: word;
begin
  S.Write(CaptionBarHeight);
  S.Write(tabwidth);
  S.Write(plusx);
  S.Write(plusy);
  S.S.Write(TextColor, SizeOf(TColor));
  _high := High(Frames);
  S.Write(_high);
  for i := 0 to _high do
    Frames[i].Window.SaveToStream(S);
end;
procedure TPopUpMenu.LoadFromStream_Individual(S: TBinaryFile);
var
```

```
  i: integer;
  _high: integer;
  str: string;
begin
  S.S.Read(TextColor, SizeOf(TColor));
  S.Read(plusx);
  S.Read(plusy);
  S.Read(_high);
  //setlength(PopUpMenuItems, _high+1);
  S.Read(PopupMenuItems[0].Caption);
  for i := 1 to _high do
  begin
    S.Read(str);
    AddPopUpItem(str);
  end;
  opened := false;
  whichchecked := 0;
end;
procedure TPopUpMenu.SaveToStream_Individual(S: TBinaryFile);
var
  i: integer;
  _high: integer;
begin
  S.S.Write(TextColor, SizeOf(TColor));
  S.Write(plusx);
  S.Write(plusy);
  _high := High(PopUpMenuItems);
  S.Write(_high);
  for i := 0 to _high do
    S.Write(PopupMenuItems[i].Caption);
end;
procedure TCombobox.LoadFromStream_Individual(S: TBinaryFile);
var
  i: integer;
  _high: integer;
  str: string;
begin
  S.S.Read(TextColor, SizeOf(TColor));
  S.Read(plusx);
  S.Read(plusy);
  S.Read(Space);
  S.Read(_high);
  //setlength(PopUpMenuItems, _high+1);
  S.Read(ComboboxItems[0].Caption);
  for i := 1 to _high do
  begin
    S.Read(str);              //ComboboxItems[i].Caption
    AddComboBoxItem(str);
    //ComboboxItems[i].Parent := Parent;
  end;

  Opened := false;
```

```
  Whichchecked := 0;
  mouseon := 0;
  Caption := ComboboxItems[0].Caption;
end;
procedure TCombobox.SaveToStream_Individual(S: TBinaryFile);
var
  i: integer;
  _high: integer;
begin
  S.S.Write(TextColor, SizeOf(TColor));
  S.Write(plusx);
  S.Write(plusy);
  S.Write(Space);
  _high := High(ComboboxItems);
  S.Write(_high);
  for i := 0 to _high do
    S.Write(ComboboxItems[i].Caption);
end;
procedure TText.LoadFromStream_Individual(S: TBinaryFile);
var
  str: string;
begin
  S.S.Read(Color, SizeOf(TColor));
  S.Read(str);
  Text := str;
end;
procedure TText.SaveToStream_Individual(S: TBinaryFile);
begin
  S.S.Write(Color, SizeOf(TColor));
  S.Write(Text);
end;
procedure TProgressBar.LoadFromStream_Individual(S: TBinaryFile);
begin
  S.Read(Progress);
end;
procedure TProgressBar.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Progress);
end;
procedure TEdit.LoadFromStream_Individual(S: TBinaryFile);
begin
  S.Read(Text);
  S.S.Read(TextColor, SizeOf(TColor));
  S.Read(plusx);
  S.Read(plusy);

  MousePosition := length(Text) - 1;
end;
procedure TEdit.SaveToStream_Individual(S: TBinaryFile);
begin
  S.Write(Text);
  S.S.Write(TextColor, SizeOf(TColor));
```

```
  S.Write(plusx);
  S.Write(plusy);
end;

//Laden des Windows.
procedure TWindow.LoadFromStream_Individual(S: TBinaryFile);
var
  i,j: integer;
  _high: word;
  hasFrames: boolean;
begin
  S.Read(Child);
  S.Read(graphicpath);
  S.S.Read(Color, SizeOf(TColor));
  S.S.Read(FontColor, SizeOf(TColor));
  S.Read(background);
  S.Read(CaptionBar);
  S.Read(CaptionBarHeight);
  S.Read(Caption);
  S.Read(hasChilds);
  S.Read(hasFrames);


  S.Read(_high);
  for i := 1 to _high do   //absichtlich 1
    AddRadioButtonGroup;
  //_high := High(RadioButtonGroups);
  {for i := 0 to High(RadioButtonGroups) do
  begin
    RadioButtons[i].
    //RadioButtonGroups[i].SaveToStream(S);
  end;    }

  S.Read(_high);
  for i := 0 to _high do
  begin
    AddButton(0,0,0,0,'');
    Buttons[i].LoadFromStream(S);
    Buttons[i].LoadFromStream_Individual(S);
  end;

  S.Read(_high);
  for i := 0 to _high do
  begin
    AddText(0,0,'',cWhite);
    Text[i].LoadFromStream(S);
    Text[i].LoadFromStream_Individual(S);
  end;

  S.Read(_high);
  for i := 0 to _high do
  begin
```

```
  AddPanel(0,0,0,0);
  Panels[i].LoadFromStream(S);
  Panels[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddProgressBar(0,0,0,0);
  ProgressBars[i].LoadFromStream(S);
  ProgressBars[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddEdit(0,0,0,0,'');
  Edits[i].LoadFromStream(S);
  Edits[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddCheckbox(0,0,0,0,false);
  Checkboxes[i].LoadFromStream(S);
  Checkboxes[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddRadioButton(0,0,0,0,0,false);
  RadioButtons[i].LoadFromStream(S);
  RadioButtons[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddImage(0,0,0,0,false,0,0,'');
  Images[i].LoadFromStream(S);
  Images[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddCombobox(0,0,0,0,'');
  Comboboxes[i].LoadFromStream(S);
  Comboboxes[i].LoadFromStream_Individual(S);
end;
```

```
S.Read(_high);
for i := 0 to _high do
begin
  AddTextField(0,0,0,0,'');
  TextFields[i].LoadFromStream(S);
  TextFields[i].LoadFromStream_Individual(S);
end;

S.Read(_high);
for i := 0 to _high do
begin
  AddEditField(0,0,0,0,'');
  EditFields[i].LoadFromStream(S);
  EditFields[i].LoadFromStream_Individual(S);
end;

if hasFrames then
begin
  S.Read(_high);
  for i := 0 to _high do
  begin
    j := GUIclass.AddWindow(0,0,0,0,0,'');
    AddFrames(0,0,0,0,Guiclass.Windows[j]);
    Frames[i].LoadFromStream(S);
    Frames[i].LoadFromStream_Individual(S);
  end;
end;

if hasChilds then
begin
  S.Read(_high);
  for i := 0 to _high do
  begin
    j := GUIclass.AddWindow(0,0,0,0,0,'');
    ChildWindows[i].Child := Guiclass.Windows[j];
    ChildWindows[i].Child.LoadFromStream(S);
  end;
end;

LoadTexture(graphicpath,graphic,false);
hasonmouseover := false;
//oldPoint:         TPoint;
//oldWidthHeight:   TPoint;

MinimizeButton.LoadFromStream(S);
MaximizeButton.LoadFromStream(S);
CloseButton.LoadFromStream(S);

MinimizeButton.pressed := false;
MinimizeButton.minimized := false;
MaximizeButton.maximized := false;
```

```
  MaximizeButton.pressed := false;
  CloseButton.pressed := false;
end;

//Speichern des Windows.
procedure TWindow.SaveToStream_Individual(S: TBinaryFile);
var
  i: integer;
  _high: word;
  hasFrames: boolean;
begin
  if Frames[0].Used then
    hasFrames := true
  else
    hasFrames := false;
  S.Write(Child);
  S.Write(graphicpath);
  S.S.Write(Color, SizeOf(TColor));
  S.S.Write(FontColor, SizeOf(TColor));
  S.Write(background);
  S.Write(CaptionBar);
  S.Write(CaptionBarHeight);
  S.Write(Caption);
  S.Write(hasChilds);
  S.Write(hasFrames);

  _high := High(RadioButtonGroups);
  S.Write(_high);
  {for i := 0 to High(RadioButtonGroups) do
  begin
    RadioButtons[i].
    //RadioButtonGroups[i].SaveToStream(S);
  end;    }


  _high := High(buttons);
  S.Write(_high);
  for i := 0 to High(buttons) do
  begin
    Buttons[i].SaveToStream(S);
    Buttons[i].SaveToStream_Individual(S);
  end;

  _high := High(Text);
  S.Write(_high);
  for i := 0 to High(Text) do
  begin
    Text[i].SaveToStream(S);
    Text[i].SaveToStream_Individual(S);
  end;

  _high := High(Panels);
```

```
  S.Write(_high);
  for i := 0 to High(Panels) do
  begin
    Panels[i].SaveToStream(S);
    Panels[i].SaveToStream_Individual(S);
  end;

  _high := High(ProgressBars);
  S.Write(_high);
  for i := 0 to High(ProgressBars) do
  begin
    ProgressBars[i].SaveToStream(S);
    ProgressBars[i].SaveToStream_Individual(S);
  end;

  _high := High(Edits);
  S.Write(_high);
  for i := 0 to High(Edits) do
  begin
    Edits[i].SaveToStream(S);
    Edits[i].SaveToStream_Individual(S);
  end;

  _high := High(Checkboxes);
  S.Write(_high);
  for i := 0 to High(Checkboxes) do
  begin
    Checkboxes[i].SaveToStream(S);
    Checkboxes[i].SaveToStream_Individual(S);
  end;

  _high := High(RadioButtons);
  S.Write(_high);
  for i := 0 to High(RadioButtons) do
  begin
    RadioButtons[i].SaveToStream(S);
    RadioButtons[i].SaveToStream_Individual(S);
  end;

  _high := High(Images);
  S.Write(_high);
  for i := 0 to High(Images) do
  begin
    Images[i].SaveToStream(S);
    Images[i].SaveToStream_Individual(S);
  end;

  _high := High(Comboboxes);
  S.Write(_high);
  for i := 0 to High(Comboboxes) do
  begin
    Comboboxes[i].SaveToStream(S);
```

```
    Comboboxes[i].SaveToStream_Individual(S);
  end;

  _high := High(TextFields);
  S.Write(_high);
  for i := 0 to High(TextFields) do
  begin
    TextFields[i].SaveToStream(S);
    TextFields[i].SaveToStream_Individual(S);
  end;

  _high := High(EditFields);
  S.Write(_high);
  for i := 0 to High(EditFields) do
  begin
    EditFields[i].SaveToStream(S);
    EditFields[i].SaveToStream_Individual(S);
  end;

  if hasFrames then
  begin
    _high := High(Frames);
    S.Write(_high);
    for i := 0 to High(Frames) do
    begin
      Frames[i].SaveToStream(S);
      Frames[i].SaveToStream_Individual(S);
    end;
  end;

  if hasChilds then
  begin
    _high := High(ChildWindows);
    S.Write(_high);
    for i := 0 to High(ChildWindows) do
    begin
       ChildWindows[i].Child.SaveToStream(S);
    end;
  end;
  MinimizeButton.SaveToStream(S);
  MaximizeButton.SaveToStream(S);
  CloseButton.SaveToStream(S);
end;

function TGUIclass.LoadWindow(path: string; var Window: array of integer): boolean;
var
  s: TBinaryFile;
  i,j: integer;
  //rec: TSaveRecord;
  w: word;
begin
  result := true;
```

```
  try
    s := TBinaryFile.Create(path,false);
    s.ReadHeader;
    s.Read(w);
    for i := 0 to w do
    begin
      j := AddWindow(0,0,0,0,0,'');
      if high(Window) >= i then
        Window[i] := j;
      Windows[j].LoadFromStream(S);
    end;
  except
    result := false;
  end;
  s.CloseFile;
end;

function TGUIclass.SaveWindow(path: string; const Window: array of integer): boolean;
var
  s: TBinaryFile;
  i: word;
begin
  result := true;
  try
    s := TBinaryFile.Create(path,true);
    s.WriteHeader(VERSION, 'GUI');
    i := High(Window);
    S.Write(i);
    for i := 0 to High(Window) do
      Windows[Window[i]].SaveToStream(S);
  except
    result := false;
  end;
  s.CloseFile;
end;

{-------------------------------------------------------------------------------}
{-------- Rendern und Deininitialisierung/Initialisierung ----------------------}
{-------------------------------------------------------------------------------}
procedure TGUIclass.Render;
var
  j: integer;
begin
  //Update der sich ändernden Elemente:
  //inc(Frame);
  //if frame mod 60 = 0 then self.showcursor := not showcursor;

  glEnable(GL_TEXTURE_2D);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

  //Ausgabe der Schriften
```

```
  PrintTextarray;

  //Ausgabe der Fenster
  for j := 0 to High(Windows) do
    if not Windows[j].Child then
    begin
      Windows[j].Render;
    end;

  glColor4f(1,1,1,1);

  //glDisable(GL_TEXTURE_2D);
  //glDisable(GL_BLEND);
end;

procedure TWindow.Render;
var
  i: integer;
{
    minimize:         cardinal;
    minimize_m:       cardinal;
    normalize:        cardinal;
    normalize_m:      cardinal;
    maximize:         cardinal;
    maximize_m:       cardinal;
    close:            cardinal;
    close_m:          cardinal;        }
begin
  if Visible then
  begin
    glColor4f(Color.R, Color.G, Color.B, Color.A);
    if Captionbar then
    begin
      //Render the Background
      if background and not MinimizeButton.minimized then
      begin
        glBindTexture(GL_TEXTURE_2D, graphic);
        glRenderQuad(x, y, x+Width, y+Height, z);
      end;

      //Render the CaptionBar
      glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].captionbar);
      glRenderQuad(x+CaptionBarHeight,y, x+Width-CaptionBarHeight, y+CaptionBarHeight,
z+1.45, 0.125,0,0.875,1);
      glRenderQuad(x,y, x+CaptionBarHeight, y+CaptionBarHeight, z+1.45,0,0,0.125,1);
      glRenderQuad(x+Width-CaptionBarHeight,y, x+Width, y+CaptionBarHeight,
z+1.45,0.875,0,1,1);

      //render the minimize, maximize, close buttons...
      if minimizebutton.Visible then
      begin
        if minimizebutton.pressed then
```

```
        glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].minimize_m)
      else
        glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].minimize);
      glRenderQuad(x+minimizebutton.X,y+minimizebutton.Y,x+minimizebutton.X+minimizebutto
n.width,y+minimizebutton.Y+minimizebutton.height,z+1.47);
    end;

    if MaximizeButton.Visible then
    begin
      if MaximizeButton.pressed then
        if MaximizeButton.maximized then
          glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].normalize_m)
        else
          glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].maximize_m)
      else
        if MaximizeButton.maximized then
          glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].normalize)
        else
          glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].maximize);
      glRenderQuad(x+MaximizeButton.X,y+MaximizeButton.Y,x+MaximizeButton.X+MaximizeBut
ton.width,y+MaximizeButton.Y+MaximizeButton.height,z+1.47);
    end;

    if CloseButton.Visible then
    begin
      if closebutton.pressed then
        glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].close_m)
      else
        glBindTexture(GL_TEXTURE_2D, GUIclass.Skins[Skin].close);
      glRenderQuad(x+closebutton.X,y+closebutton.Y,x+closebutton.X+closebutton.width,y+close
button.Y+closebutton.height,z+1.47);
    end;

  end else
  begin
    if background and not MinimizeButton.minimized then
    begin
      glBindTexture(GL_TEXTURE_2D, graphic);
      glRenderQuad(x, y, x+Width, y+Height, z);
    end;
  end;

  if not MinimizeButton.minimized then
  begin
    // draw the Panels
    for i :=0 to High(Panels) do
      if Panels[i].Visible then
        Panels[i].Render;

    // draw the ProgressBars
    for i :=0 to High(ProgressBars) do
      if ProgressBars[i].Visible then
```

```
    ProgressBars[i].Render;

// draw the Edits
for i :=0 to High(Edits) do
    if Edits[i].Visible then
      Edits[i].Render;

// draw the Buttons
for i :=0 to High(Buttons) do
  if Buttons[i].Visible then
    Buttons[i].Render;

// draw the Checkboxes
for i :=0 to High(Checkboxes) do
  if Checkboxes[i].Visible then
    Checkboxes[i].Render;

// draw the RadioButtons
for i :=0 to High(RadioButtons) do
  if RadioButtons[i].Visible then
    RadioButtons[i].Render;

// draw the TextFields
for i :=0 to High(TextFields) do
  if TextFields[i].Visible then
    TextFields[i].Render;

// draw the EditFields
for i :=0 to High(EditFields) do
  if EditFields[i].Visible then
    EditFields[i].Render;

// draw the Images
for i :=0 to High(Images) do
  if Images[i].Visible then
    Images[i].Render;

// draw the ComboBoxes
for i :=0 to High(ComboBoxes) do
  if Comboboxes[i].Visible then
    Comboboxes[i].Render;

//PopUpMenus
for i := 0 to High(PopUpMenus) do
  if PopUpMenus[i].opened and PopUpMenus[i].Visible then
    PopUpMenus[i].Render;

//Frames
for i := 0 to High(Frames) do
  if Frames[i].Visible then
    Frames[i].Render;
```

```
      // draw the Text
      for i :=0 to High(Text) do
        if Text[i].Visible then
        begin
          glColor4f(Text[i].color.r,Text[i].color.g,Text[i].color.b,color.a);
          GUIclass.PrintFont(X+Text[i].X, Y + Text[i].Y,Text[i].Z,Text[i].Font,PChar(Text[i].Text));
        end;
    end;
    glColor4f(FontColor.r,FontColor.g,FontColor.b,Color.a);
    if captionbar then GUIclass.PrintFont(X+11, Y+5, z+1.47, Font, PChar(Caption));

    // draw the child windows
    if hasChilds then
      for i :=0 to High(ChildWindows) do
      begin
        glPushMatrix;
          gltranslatef(0,0, 1.5 + i/10);
          ChildWindows[i].Child.Render;
        glPopMatrix
      end;
  end;
end;

procedure TGUIclass.GoOrtho(Width, Height, zNear, zFar: double);
begin
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  // In orthagonale (2D) Ansicht wechseln
  glOrtho(0,Width,Height,0,zNear,zFar);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity;
end;

procedure TGUIclass.ExitOrtho(FOV, aspect, znear, zfar: double);
begin
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  // Perspective, FOV und Tiefenreichweite setzen
  gluPerspective(FOV, aspect, znear, zfar);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity;
end;

procedure TWindow.FreeComponents;
var
  i,j: integer;
begin
  for i := 0 to High(Buttons) do
    Buttons[i].Free;

  for i := 0 to High(Text) do
    Text[i].Free;
```

```
for i := 0 to High(Panels) do
  Panels[i].Free;

for i := 0 to High(RadioButtonGroups) do
  RadioButtonGroups[i].Free;

for i := 0 to High(RadioButtons) do
  RadioButtons[i].Free;

for i := 0 to High(Checkboxes) do
  Checkboxes[i].Free;

for i := 0 to High(Edits) do
  Edits[i].Free;

for i := 0 to High(ProgressBars) do
  ProgressBars[i].Free;

for i := 0 to High(TextFields) do
  TextFields[i].Free;

for i := 0 to High(EditFields) do
  EditFields[i].Free;

for i := 0 to High(PopUpMenus) do
begin
  for j := 0 to High(PopUpMenus[i].PopUpMenuItems) do
    PopUpMenus[i].PopUpMenuItems[j].Free;
  PopUpMenus[i].Free;
end;

for i := 0 to High(ComboBoxes) do
begin
  for j := 0 to High(Comboboxes[i].ComboboxItems) do
    Comboboxes[i].ComboboxItems[j].Free;
  ComboBoxes[i].Free;
end;

for i := 0 to High(Images) do
  Images[i].Free;

for i := 0 to High(Frames) do
begin
  if Frames[i].used then
    for j := 0 to High(Frames[i].Frames) do
    begin
      Frames[i].Frames[j].Window.FreeComponents;
      Frames[i].Frames[j].Window.Free;
    end;
  Frames[i].Free;
end;
```

```
  if hasChilds then
  begin
    for i := 0 to High(ChildWindows) do
    begin
      ChildWindows[i].Child.FreeComponents;
      ChildWindows[i].Child.Free;
    end;
  end;
end;

procedure TGUIclass.Init;
begin
  setlength(GUIclass.Windows, 1);
  setlength(GUIclass.Text,    1);
  setlength(GUIclass.Fonts,   1);
  setlength(GUIclass.Skins,   1);

  GUIclass.Windows[0]        :=  TWindow.Create;
  GUIclass.Text[0]           :=  TText.Create;

  with GUIclass.Windows[0] do
  begin
    setlength(Buttons,        1);
    setlength(Text,           1);
    setlength(RadioButtons,   1);
    setlength(Checkboxes,     1);
    setlength(Edits,          1);
    setlength(Panels,         1);
    setlength(ProgressBars,   1);
    setlength(PopUpMenus,     1);

    Buttons[0]         := TButton.Create;
    Text[0]            := TText.Create;
    RadioButtons[0]    := TRadioButton.Create;
    Checkboxes[0]      := TCheckbox.Create;
    Edits[0]           := TEdit.Create;
    Panels[0]          := TPanel.Create;
    ProgressBars[0]    := TProgressBar.Create;
    PopUpMenus[0]      := TPopUpMenu.Create;

    MinimizeButton     := TMinimizeButton.Create;
    MaximizeButton     := TMaximizeButton.Create;
    CloseButton        := TCloseButton.Create;
    Parent := Windows[0];
  end;

  GUIclass.Windows[0].used          := false;
  GUIclass.Fonts[0].used            := false;
  GUIclass.Text[0].used             := false;
  GUIclass.Skins[0].used            := false;
```

```
  GUIclass.Windows[0]._visible       := false;
  GUIclass.Text[0]._visible          := false;

  GUIclass.lastMouseDown := GUIclass.Windows[0];
  GUIclass.lastMouseUp := GUIclass.Windows[0];
  GUIclass.lastMouseMove := GUIclass.Windows[0];
  GUIclass.active := Guiclass.Windows[0];
end;

destructor TGUIclass.Destroy;
begin
  for _counter := 0 to High(GUIclass.Windows) do
  begin
    if not GUIclass.Windows[_counter].Child then
    begin
      GUIclass.Windows[_counter].FreeComponents;
      GUIclass.Windows[_counter].Free;
    end;
  end;

  for _counter := 0 to High(GUIclass.Text) do
    GUIclass.Text[_counter].Free;
end;

initialization
  GUIclass := TGUIclass.Create;
  GUIclass.Init;

finalization
  GUIclass.Destroy;

end.
```

9. GUIAdds

```
unit GUIAdds;

interface

uses
  GUI,
  dglOpenGl,
  Unit_SDL,
  SysUtils,
  oooal,
  UCharacters,
  UNetwork,
  ULogger,
  IniFiles,
  Basics;

type
  TGUIAdd = class(TObject)
```

```
    procedure CalcStats;
    procedure Init;
    procedure Render;
    procedure Resize;
  end;

  TConsole = class(TObject)
    Window:     integer;                //GUI number.
    EditField:  TEditField;
    Edit:       TEdit;
    TextField:  TTextField;             //for ingame messages. Displays the same like the Console.
    procedure AddString(s: string);
    procedure AddStringAndLog(Warning, Where: string);
    procedure ClearLastLine;
    constructor Create;
    destructor Destroy; override;
    procedure ProceedInput(Input: string);
  end;

var
  GUIAdd: TGUIAdd;
  Console: TConsole;

  //Fenster
  wMain, wCredits, wNewGame, wJoinGame,
   wSetup, wExit, wIngame, wChangeLevel,
   wBuy, wChat, wChars, wStatistik:  integer;
  wNetError: integer;

  //Schriften
  farial10, fbodini10,fverdana10, fGhost,
  fcouriernew13, fcouriernew12,fcouriernew9, fcouriernew24,
   fcouriernew48, fcouriernew60: integer;

  //sounds
  sclick: integer;

  tempLevelName: string;  //um ein neues level zu starten, muss vorübergehend ein string
abgespeichert werden.

  sChars: array[0..3] of string;

implementation

uses
  MainUnit, UShader, UOctree, ULevels;

const

cGUIRed: TColor = ( r:  162/255; g:  0; b:  0; a:  1 );

{
```

```
------------------------------------------------------------------------------
        GUI Events
------------------------------------------------------------------------------
}
procedure overMainForm(X,Y: integer; Element: TGUIObject);
var
  i: integer;
begin
  for i := 0 to 6 do
  begin
    GUIclass.Windows[wMain].Text[i].font  := fCourierNew12;
    GUIclass.Windows[wMain].Text[i].color := cGUIRed;
  end;
end;

procedure overStartButtons(X,Y: integer; Element: TGUIObject);
begin
  overMainForm(X,Y,Element);       //um die alten Buttons wieder "abzustellen"
  GUIclass.Windows[0].Text[Element.Index].color := cwhite;
  GUIclass.Windows[0].Text[Element.Index].font := fCourierNew13;
  OpenAL.sounds[sclick].Play;
  //evtl ton ertönen lassen.
end;

procedure onExitLevel(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  game.EndLevel;
end;

procedure onContinueLevel(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  game.ingame := true;
end;

procedure onNewGamePortChange(Key: Word; Element: TGUIObject);
begin
  if Element.Parent.Checkboxes[0].Checked then
  begin
    if Element.Index = 0 then
      Element.Parent.Edits[1].Text := Element.Parent.Edits[0].Text
    else
      Element.Parent.Edits[0].Text := Element.Parent.Edits[1].Text;
  end;
end;

procedure onNewGameButton(X,Y: integer; Element: TGUIObject; Button: integer);
var
  fname: string;
begin
  fname :=
GUIclass.Windows[wNewGame].Comboboxes[0].ComboboxItems[GUIclass.Windows[wNewGame].
Comboboxes[0].whichchecked].Caption;
```

```
  if Fileexists('data\maps\' + fname + '\map.3ds') or Fileexists('data\maps\' + fname + '\map.lvl')
then
  begin
    if (length(Element.Parent.Edits[3].Text) > 2) and (length(Element.Parent.Edits[3].Text) < 24)
then
    begin
      if isInteger(GUIclass.Windows[wNewGame].Edits[0].Text) then
      begin
        if isInteger(GUIclass.Windows[wNewGame].Edits[1].Text) then
        begin
          if isInteger(GUIclass.Windows[wNewGame].Edits[2].Text) then
          begin
            if isInteger(GUIclass.Windows[wNewGame].Edits[4].Text) then
            begin
              if length(Element.Parent.Edits[5].Text) > 2 then
              begin
                game.startlevel(fname);  //Name ändern
                Net.StartServer(strtoint(GUIclass.Windows[wNewGame].Edits[2].Text),
                                strtoint(GUIclass.Windows[wNewGame].Edits[1].Text),
                                strtoint(GUIclass.Windows[wNewGame].Edits[0].Text),
                                strtoint(GUIclass.Windows[wNewGame].Edits[4].Text),
                                GUIclass.Windows[wNewGame].Checkboxes[1].Checked,
                                GUIclass.Windows[wNewGame].Edits[5].Text
                                );
                UCharacters.Chars.char[UCharacters.Chars.ownChar].Name :=
Element.Parent.Edits[3].Text;
              end
              else
                game.onError('The Server Name must contain at least 3 letters.', 'GUIAdds');
            end
            else
              game.onError('Gold per round must be a valid Number.', 'GUIAdds');
          end
          else
            game.onError('Max. Players must be a valid Number.', 'GUIAdds');
        end
        else
          game.onError('TCP Port must be a valid Number.', 'GUIAdds');
      end
      else
        game.onError('UDP Port must be a valid Number.', 'GUIAdds');
    end
    else
      game.onError('Your Name must contain at least 3 letters and less than 24.', 'GUIAdds');
  end
  else
    game.onError('Map konnte nicht gefunden werden: ' + fname, 'GUIAdds');
end;

procedure onNewGame(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Guiclass.Windows[wNewGame].X := round(halfscreenwidth  -
```

```
Guiclass.Windows[wNewGame].Width/2);
  Guiclass.Windows[wNewGame].Y := round(halfscreenheight -
Guiclass.Windows[wNewGame].Height/2);
  Guiclass.Windows[wNewGame].Visible := true;
end;

procedure onJoinGamePortChange(Key: Word; Element: TGUIObject);
begin
  if Element.Parent.Checkboxes[0].Checked then
  begin
    if Element.Index = 1 then
      Element.Parent.Edits[2].Text := Element.Parent.Edits[1].Text
    else
      Element.Parent.Edits[1].Text := Element.Parent.Edits[2].Text;
  end;
end;

procedure onJoinGameButton(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  if isInteger(GUIclass.Windows[wJoinGame].Edits[2].Text) then
  begin
    if isInteger(GUIclass.Windows[wJoinGame].Edits[1].Text) then
    begin
      if length(Element.Parent.Edits[3].Text) > 2 then
      begin
        Net.StartClient(         GUIclass.Windows[wJoinGame].Edits[0].Text,
                        strtoint(GUIclass.Windows[wJoinGame].Edits[1].Text),
                        strtoint(GUIclass.Windows[wJoinGame].Edits[2].Text),
                                 GUIclass.Windows[wJoinGame].Edits[3].Text
                        );
      end
      else
        game.onError('Your Name must contain at least 3 letters.', 'GUIAdds');
    end
    else
      game.onError('TCP Port must be a valid Number.', 'GUIAdds');
  end
  else
    game.onError('UDP Port must be a valid Number.', 'GUIAdds');
end;

procedure onJoinGame(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Guiclass.Windows[wJoinGame].X := round(halfscreenwidth  -
Guiclass.Windows[wJoinGame].Width/2);
  Guiclass.Windows[wJoinGame].Y := round(halfscreenheight -
Guiclass.Windows[wJoinGame].Height/2);
  Guiclass.Windows[wJoinGame].Visible := true;
end;

procedure onSetupSave(X,Y: integer; Element: TGUIObject; Button: integer);
var
```

```
  Window: TWindow;
  FrameWin: TWindow;
begin
  //save..
  Window := Element.Parent;
  FrameWin := Element.Parent.Frames[0].Frames[0].Window;
  GameVariabeln.VideoConfig.ResolutionIndex:= FrameWin.Comboboxes[0].Whichchecked;
  GameVariabeln.VideoConfig.GraphicQuality := FrameWin.Comboboxes[1].Whichchecked;
  GameVariabeln.VideoConfig.BitDepthIndex  := FrameWin.Comboboxes[2].Whichchecked;
  GameVariabeln.VideoConfig.Gamma          := strtofloat(FrameWin.Edits[0].Text);
  GameVariabeln.VideoConfig.Shader         := FrameWin.Checkboxes[0].Checked;
  GameVariabeln.VideoConfig.WindowMode     := FrameWin.Checkboxes[1].Checked;
  GameVariabeln.VideoConfig.VerticalSync   := FrameWin.Checkboxes[2].Checked;

  FrameWin := Window.Frames[0].Frames[1].Window;
  GameVariabeln.AudioConfig.Sound          := FrameWin.Checkboxes[1].Checked;
  GameVariabeln.AudioConfig.Music          := FrameWin.Checkboxes[0].Checked;
  GameVariabeln.AudioConfig.SoundVolume    := FrameWin.ProgressBars[0].progress;
  GameVariabeln.AudioConfig.MusicVolume    := FrameWin.ProgressBars[1].progress;

  FrameWin := Window.Frames[0].Frames[2].Window;
  //KEYS

  FrameWin := Window.Frames[0].Frames[3].Window;
  GameVariabeln.MouseConfig.MouseSpeed     := FrameWin.ProgressBars[0].progress;
  //Acc. gerade berechnen.
  GameVariabeln.MouseConfig.MouseAcc       := sqr((FrameWin.ProgressBars[1].progress)*2);
  GameVariabeln.MouseConfig.InvertMouse    := FrameWin.Checkboxes[0].Checked;

  FrameWin := Window.Frames[0].Frames[4].Window;
  GameVariabeln.MiscConfig.ShowDebug       := FrameWin.Checkboxes[0].Checked;
  GameVariabeln.MiscConfig.Sleep           := strtoint(FrameWin.Edits[0].Text);
  GameVariabeln.MiscConfig.NearClipping    := strtofloat(FrameWin.Edits[1].Text);
  GameVariabeln.MiscConfig.FarClipping     := strtofloat(FrameWin.Edits[2].Text);

  Element.Parent.Visible := false;
end;

procedure onSetupCancel(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Element.Parent.Visible := false;
end;

procedure onSetup(X,Y: integer; Element: TGUIObject; Button: integer);
var
  Window : TWindow;
  FrameWin: TWindow;
begin
  Window := GUIclass.Windows[wSetup];

  Window.X := round(halfscreenwidth  - Window.Width/2);
  Window.Y := round(halfscreenheight - Window.Height/2);
```

```
  Window.Visible := true;

  FrameWin := Window.Frames[0].Frames[0].Window;
  FrameWin.Comboboxes[0].Whichchecked := GameVariabeln.VideoConfig.ResolutionIndex;
  FrameWin.Comboboxes[0].Mouseon      := GameVariabeln.VideoConfig.ResolutionIndex;
  FrameWin.Comboboxes[1].Whichchecked := GameVariabeln.VideoConfig.GraphicQuality;
  FrameWin.Comboboxes[1].Mouseon      := GameVariabeln.VideoConfig.GraphicQuality;
  FrameWin.Comboboxes[2].Whichchecked := GameVariabeln.VideoConfig.BitDepthIndex;
  FrameWin.Comboboxes[2].Mouseon      := GameVariabeln.VideoConfig.BitDepthIndex;
  FrameWin.Edits[0].Text          := floattostr(GameVariabeln.VideoConfig.Gamma);
  FrameWin.Checkboxes[0].Checked    := GameVariabeln.VideoConfig.Shader;
  FrameWin.Checkboxes[1].Checked    := GameVariabeln.VideoConfig.WindowMode;
  FrameWin.Checkboxes[2].Checked    := GameVariabeln.VideoConfig.VerticalSync;

  FrameWin := Window.Frames[0].Frames[1].Window;
  FrameWin.Checkboxes[1].Checked    := GameVariabeln.AudioConfig.Sound;
  FrameWin.Checkboxes[0].Checked    := GameVariabeln.AudioConfig.Music;
  FrameWin.ProgressBars[0].progress := GameVariabeln.AudioConfig.SoundVolume;
  FrameWin.ProgressBars[1].progress := GameVariabeln.AudioConfig.MusicVolume;

  FrameWin := Window.Frames[0].Frames[2].Window;
  //KEYS

  FrameWin := Window.Frames[0].Frames[3].Window;
  FrameWin.ProgressBars[0].progress := GameVariabeln.MouseConfig.MouseSpeed;
  FrameWin.ProgressBars[1].progress := sqrt(GameVariabeln.MouseConfig.MouseAcc)/2;
  FrameWin.Checkboxes[0].Checked    := GameVariabeln.MouseConfig.InvertMouse;

  FrameWin := Window.Frames[0].Frames[4].Window;
  FrameWin.Checkboxes[0].Checked    := GameVariabeln.MiscConfig.ShowDebug;
  FrameWin.Edits[0].Text          := inttostr(GameVariabeln.MiscConfig.Sleep);
  FrameWin.Edits[1].Text          := floattostr(GameVariabeln.MiscConfig.NearClipping);
  FrameWin.Edits[2].Text          := floattostr(GameVariabeln.MiscConfig.FarClipping);
end;

procedure onCredits(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Guiclass.Windows[wCredits].X := round(halfscreenwidth  -
Guiclass.Windows[wCredits].Width/2);
  Guiclass.Windows[wCredits].Y := round(halfscreenheight -
Guiclass.Windows[wCredits].Height/2);
  Guiclass.Windows[wCredits].Visible := true;
end;

procedure onExit(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Guiclass.Windows[wExit].X := halfscreenwidth - (Guiclass.Windows[wExit].Width div 2);
  Guiclass.Windows[wExit].Y := halfscreenheight - (Guiclass.Windows[wExit].Height div 2);
  Guiclass.Windows[wExit].Visible := true;
end;

procedure StopGame(X,Y: integer; Element: TGUIObject; Button: integer);
```

```
begin
  done := -1;
end;

procedure ChangeLevel(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  game.EndLevel;
  game.initlevel(tempLevelName);
end;

procedure MakeBuyItemsInvis;
var
  i: integer;
begin
  with GUIclass.Windows[wBuy] do
  begin
    for i := 3 to High(Images) do   //es gibt schon 3 bilder, die benützt werden, darum fängt das
ganze erst bei 3 an
      Images[i].Visible := false;
    for i := 0 to 29 do
      Text[i].Visible := false;
    for i := 0 to 14 do
      TextFields[i].Visible := false;
  end;
end;

procedure onEquip(X,Y: integer; Element: TGUIObject; Button: integer);
var
  i: integer;
begin
  MakeBuyItemsInvis;
  for i := 13 to 17 do
    GUIclass.Windows[wBuy].Images[i].visible := true;
  for i := 20 to 29 do
    GUIclass.Windows[wBuy].Text[i].Visible := true;
  for i := 10 to 14 do
    GUiclass.Windows[wBuy].TextFields[i].Visible := true;
end;

procedure onPistol(X,Y: integer; Element: TGUIObject; Button: integer);
var
  i: integer;
begin
  MakeBuyItemsInvis;
  for i := 3 to 7 do
    GUIclass.Windows[wBuy].Images[i].visible := true;
  for i := 0 to 9 do
    GUIclass.Windows[wBuy].Text[i].Visible := true;
  for i := 0 to 4 do
    GUiclass.Windows[wBuy].TextFields[i].Visible := true;
end;
```

```
procedure onRifle(X,Y: integer; Element: TGUIObject; Button: integer);
var
  i: integer;
begin
  MakeBuyItemsInvis;
  for i := 8 to 12 do
    GUIclass.Windows[wBuy].Images[i].Visible := true;
  for i := 10 to 19 do
    GUIclass.Windows[wBuy].Text[i].Visible := true;
  for i := 5 to 9 do
    GUiclass.Windows[wBuy].TextFields[i].Visible := true;
end;

procedure onPistolItem(X,Y: integer; Element: TGUIObject; Button: integer); //= buy pistol
begin
  Chars.CurrentChar.buyPistol(Element.tag);
end;

procedure onRifleItem(X,Y: integer; Element: TGUIObject; Button: integer);  //= buy rifle
begin
  Chars.CurrentChar.buyRifle(Element.tag);
end;

procedure onEquipItem(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Chars.CurrentChar.buyEquip(Element.tag);
end;

procedure CloseBuyWindow(Element: TGUIObject);
begin
  game.ShowCursor := false;
end;

procedure onCloseChangeLastMouseUp(Element: TGUIObject);
begin
  GUIclass.lastMouseUp := GUIclass.Windows[0];
end;

procedure onCloseConsole(Element: TGUIObject);
begin
  Element.Parent.Edits[0].Text := '';
  GUIclass.lastMouseUp := GUIclass.Windows[0];
end;

procedure onChatCancel(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Element.Parent.Visible := false;
  game.ShowCursor := false;
end;

procedure onChatSend(X,Y: integer; Element: TGUIObject; Button: integer);
begin
```

```
  Net.Chat(Element.Parent.Edits[0].Text, Element.Parent.Checkboxes[0].Checked);
  Element.Parent.Visible := false;
  game.ShowCursor := false;
end;

procedure onChatChange(Key: Word; Element: TGUIObject);
begin
  if Length(Element.Parent.Edits[0].Text) > 50 then
    setlength(Element.Parent.Edits[0].Text, 50);
end;

procedure onCharImage(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  Chars.char[Chars.ownChar].ModelNr := Element.tag;
  Element.Parent.Visible := false;
end;


{
------------------------------------------------------------------------------
       TGUIAdd
------------------------------------------------------------------------------
}

procedure TGUIAdd.CalcStats;
var
  terarr, ctarr: array of integer;
  i,j, c1,c2 : integer;
  madechange: boolean;
  temp: integer;
begin
  setlength(ctarr, Chars.numCTs);
  setlength(terarr, Chars.numTerrors);

  //fill data in the arrays.
  c1 := 0;
  c2 := 0;
  for i := 0 to Chars.numChars - 1 do
    if Chars.char[i].Terrorist then
    begin
      terarr[c1] := i;
      inc(c1);
    end else
    begin
      ctarr[c2] := i;
      inc(c2);
    end;

  //sort CTs
  madechange := true;
  while madechange do
  begin
```

```
  madechange := false;
  for i := 0 to Chars.numCTs - 2 do
    if Chars.char[ctarr[i]].Kills < Chars.char[ctarr[i+1]].Kills then
    begin
      temp := ctarr[i];
      ctarr[i] := ctarr[i + 1];
      ctarr[i+1] := temp;
      madechange := true;
    end;
end;

madechange := true;
//sort Terrorists
while madechange do
begin
  madechange := false;
  for i := 0 to Chars.numTerrors - 2 do
    if Chars.char[terarr[i]].Kills < Chars.char[terarr[i+1]].Kills then
    begin
      temp := terarr[i];
      terarr[i] := terarr[i + 1];
      terarr[i+1] := temp;
      madechange := true;
    end;
end;

//"print" the lists into the "Statistik"-Window
with GUIclass.Windows[wStatistik] do
begin
  for i := 0 to 5 do
  begin
    TextFields[i].Text := '';
  end;

  j := Chars.numTerrors;
  setlength(TextFields[0].Stringlist, j);
  setlength(TextFields[1].Stringlist, j);
  setlength(TextFields[2].Stringlist, j);
  for i := 0 to Chars.numTerrors - 1 do
  begin
    TextFields[0].Stringlist[i] := Chars.char[terarr[i]].name;
    TextFields[1].Stringlist[i] := inttostr(Chars.char[terarr[i]].Kills);
    TextFields[2].Stringlist[i] := inttostr(Chars.char[terarr[i]].Deaths);
  end;

  j := Chars.numCTs;
  setlength(TextFields[3].Stringlist, j);
  setlength(TextFields[4].Stringlist, j);
  setlength(TextFields[5].Stringlist, j);
  for i := 0 to Chars.numCTs - 1 do
  begin
    TextFields[3].Stringlist[i] := Chars.char[ctarr[i]].name;
```

```
      TextFields[4].Stringlist[i] := inttostr(Chars.char[ctarr[i]].Kills);
      TextFields[5].Stringlist[i] := inttostr(Chars.char[ctarr[i]].Deaths);
    end;
  end;
end;

procedure TGUIAdd.Init;
const
  _g = 15;
var
  a: array of integer;
  i,j: integer;
  imgwidth: integer;
  ini: TInifile;
begin
  farial10      := GUIclass.AddFont('data\fonts\arial10.fnt');
  fbodini10     := GUIclass.AddFont('data\fonts\bodini10.fnt');
  fverdana10    := GUIclass.AddFont('data\fonts\verdana12.fci', 0.6);
  fcouriernew13 := GUIclass.AddFont('data\fonts\couriernew48.fci', 0.5);
  fcouriernew9  := GUIclass.AddFont('data\fonts\couriernew48.fci', 0.35);
  fcouriernew24 := GUiclass.AddFont('data\fonts\couriernew48.fci', 1);
  fcouriernew12 := GUIclass.AddFont('data\fonts\couriernew48.fci', 0.45);
  fcouriernew48 := GUIclass.AddFont('data\fonts\couriernew48.fci', 2);
  fcouriernew60 := GUIclass.AddFont('data\fonts\couriernew48.fci', 2.5);
  fGhost        := GUIclass.AddFont('data\fonts\brokenghost36.fci', 1);

  GUIclass.AddSkin('data\skins\windows\');
  GUIclass.AddSkin('data\skins\black blood\');

  wMain := GUIclass.AddWindow(0,0,Screen_Width, Screen_Height, 0, 'Main-Window',
'data\GUI\main.jpg');
  with GUIclass.Windows[wMain] do
  begin
    Z := -9;
    Visible := true;
    font := fcouriernew12;
    onMouseOver := overMainForm;
    AddText(30, Screen_Height - 110, 'New Game', cGUIRed);       Text[0].onClick :=
onNewGame;       Text[0].onMouseOver := overStartButtons;
    AddText(30, Screen_Height -  90, 'Join Game', cGUIRed);      Text[1].onClick := onJoinGame;
Text[1].onMouseOver := overStartButtons;
    AddText(30, Screen_Height -  70, 'Setup', cGUIRed);          Text[2].onClick := onSetup;
Text[2].onMouseOver := overStartButtons;
    AddText(30, Screen_Height -  50, 'Credits', cGUIRed);        Text[3].onClick := onCredits;
Text[3].onMouseOver := overStartButtons;
    AddText(30, Screen_Height -  30, 'Exit', cGUIRed);           Text[4].onClick := onExit;
Text[4].onMouseOver := overStartButtons;
    AddText(30, Screen_Height - 190, 'Continue Level', cGUIRed); Text[5].onClick :=
onContinueLevel; Text[5].onMouseOver := overStartButtons;
    AddText(30, Screen_Height - 170, 'Exit Level', cGUIRed);     Text[6].onClick := onExitLevel;
Text[6].onMouseOver := overStartButtons;
    //AddText(30, 30, 'Terrorist''s Revenge', cWhite);        Text[7].font := fcouriernew48;
```

```
    sClick := openal.addsound('data\sound\click.wav');
  end;

  wIngame := GUIclass.AddWindow(0,0,Screen_Width,Screen_Height, 1, 'Ingame-Form');
  with GUIclass.Windows[wIngame] do
  begin
    Z := -12;
    AddImage(halfscreenwidth-_g, halfscreenheight-_g, 2*_g, 2*_g,true, GL_SRC_ALPHA,
GL_ONE, 'data\textures\cross3.bmp');
    AddImage((Screen_Width div 4)  -100, Screen_Height-90, 50, 50,true, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA, 'data\textures\Armor.tga');
    AddImage((Screen_Width div 4)*2-100, Screen_Height-90, 50, 50,true, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA, 'data\textures\Life.tga');
    AddImage((Screen_Width div 4)*3-100, Screen_Height-90, 50, 50,true, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA, 'data\textures\Dollar.tga');
    AddImage(Screen_Width - 70, 20,                    50, 50,true, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA, 'data\textures\magazine.tga');
    AddImage(Screen_Width - 70, 90,                    50, 50,true, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA, 'data\textures\Ammo.tga');
    background := false;
    captionbar := false;
  end;

  //Das Laden der Main-Windows.
  setlength(a, 6);
  GUIclass.LoadWindow('data\GUI\MainWindows.gui',a);
  wNewGame      := a[0];
  wJoinGame     := a[1];
  wSetup        := a[2];
  wCredits      := a[3];
  wExit         := a[4];
  wChangeLevel  := a[5];
  GUIclass.Windows[wNewGame].Visible      := false;
  GUIclass.Windows[wJoinGame].Visible     := false;
  GUIclass.Windows[wSetup].Visible        := false;
  GUIclass.Windows[wCredits].Visible      := false;
  GUIclass.Windows[wExit].Visible         := false;
  GUIclass.Windows[wChangeLevel].Visible  := false;
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wNewGame]);
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wJoinGame]);
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wSetup]);
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wCredits]);
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wExit]);
  GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wChangeLevel]);

  //Einstellungen wNewGame
  with GUIclass.Windows[wNewGame] do
  begin
    Buttons[0].onClick := onNewGameButton;
    onDestroy := onCloseChangeLastMouseUp;
    try
      ini := TIniFile.Create('data\maps\config.ini');
```

```
      j   := ini.ReadInteger('config', 'maps', 0);
      for i := 0 to j - 1 do
      begin
        if i <> 0 then
          GUIclass.Windows[wNewGame].Comboboxes[0].AddComboBoxItem(ini.ReadString('config',
'map'+inttostr(i), 'Test'))
        else
          GUIclass.Windows[wNewGame].Comboboxes[0].ComboboxItems[0].Caption :=
ini.ReadString('config', 'map'+inttostr(i), 'Test');
      end;
      Edits[0].onKeyUp := onNewGamePortChange;
      Edits[1].onKeyUp := onNewGamePortChange;
    finally
      ini.Free;
    end;

  end;

  //Einstellungen wJoinGame
  with GUIclass.Windows[wJoinGame] do
  begin
    Buttons[0].onClick := onJoinGameButton;
    Edits[1].onKeyUp := onJoinGamePortChange;
    Edits[2].onKeyUp := onJoinGamePortChange;
    onDestroy := onCloseChangeLastMouseUp;
  end;

  //Einstellungen wSetup
  with GUIclass.Windows[wSetup] do
  begin
    Buttons[0].onClick := onSetupCancel;
    Buttons[1].onClick := onSetupSave;
    onDestroy := onCloseChangeLastMouseUp;

    Frames[0].Frames[3].Window.ProgressBars[0].onMouseDown := MousePostoProgressbar;
    Frames[0].Frames[3].Window.ProgressBars[1].onMouseDown := MousePostoProgressbar;
  end;


  //einstellungen Exit:
  GUIclass.Windows[wExit].Buttons[0].onClick := StopGame;
  GUIclass.Windows[wExit].Buttons[1].onClick := CloseWindow;

  //Einstellungen ChangeLevel
  GUIclass.Windows[wChangeLevel].Buttons[1].onClick := CloseWindow;
  GUIclass.Windows[wChangeLevel].Buttons[0].onClick := ChangeLevel;

  //Windows NetError:
  wNetError := GUIclass.AddWindow(halfscreenwidth, halfscreenheight, 300,100,1, 'Network
Error');
  GUIclass.Windows[wNetError].AddText(20,50,'', cWhite);
  GUIclass.Windows[wNetError].font := fCouriernew13;
```

```
GUIclass.Windows[wNetError].CaptionBar := true;
GUIclass.Windows[wNetError].MinimizeButton.Visible := true;
GUIclass.Windows[wNetError].MaximizeButton.Visible := true;
GUIclass.Windows[wNetError].CloseButton.Visible    := true;
GUIclass.Windows[wNetError].dragevent              := true;
GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[wNetError]);


//Konsole starten
Console := TConsole.Create;

//    .....ingame fenster.....

//buy Windows:
GUIclass.LoadWindow('data\GUI\buy.gui',a);
wBuy := a[0];
GUIclass.Windows[wIngame].AddChildWindow(GUIclass.Windows[wBuy]);

//Statistik Window
GUIclass.LoadWindow('data\GUI\Statistik.gui',a);
wStatistik  := a[0];
GUIclass.Windows[wStatistik].Visible := false;
GUIclass.Windows[wIngame].AddChildWindow(GUIclass.Windows[wStatistik]);

//Chat Window
GUIclass.LoadWindow('data\GUI\Chat.gui', a);
wChat := a[0];
with GUIclass.Windows[wChat] do
begin
  Edits[0].onKeyUp := onChatChange;
  Buttons[0].onClick := onChatSend;
  Buttons[1].onClick := onChatCancel;
  Visible := false;
end;
GUIclass.Windows[wIngame].AddChildWindow(GUIclass.Windows[wChat]);

//choose character window:
GUIclass.LoadWindow('data\GUI\choosecharacter.gui',a);
wChars := a[0];
GUIclass.Windows[wIngame].AddChildWindow(GUIclass.Windows[wChars]);
with GUIclass.Windows[wChars] do
begin
  try
    ini := TINIFile.Create('data\characters\config.ini');
    sChars[0] := ini.ReadString('counter', 'counter1', 'civil');
    sChars[1] := ini.ReadString('counter', 'counter2', 'civil');
    sChars[2] := ini.ReadString('terror' , 'terror1' , 'blackman');
    sChars[3] := ini.ReadString('terror' , 'terror2' , 'blackman');
  finally
    ini.Free
  end;
  AddImage(Panels[1].X+10,Panels[1].Y+25, Panels[1].Width - 20,Panels[1].Width -
```

```
20,false,0,0, 'data\characters\' + sChars[0] + '\graphic.jpg');
   AddImage(Panels[1].X+10,Panels[1].Y+25+Panels[1].Width - 20 + 10, Panels[1].Width -
20,Panels[1].Width - 20,false,0,0, 'data\characters\' + sChars[1] + '\graphic.jpg');
   AddImage(Panels[0].X+10,Panels[0].Y+25, Panels[0].Width - 20,Panels[0].Width -
20,false,0,0, 'data\characters\' + sChars[2] + '\graphic.jpg');
   AddImage(Panels[0].X+10,Panels[0].Y+25+Panels[0].Width - 20 + 10, Panels[0].Width -
20,Panels[0].Width - 20,false,0,0, 'data\characters\' + sChars[3] + '\graphic.jpg');

   Images[0].tag := 0;
   Images[1].tag := 1;

   Images[2].tag := 10;
   Images[3].tag := 11;

   Visible := false;

   onDestroy := CloseBuyWindow;
   Images[0].onClick := onCharImage;
   Images[1].onClick := onCharImage;
   Images[2].onClick := onCharImage;
   Images[3].onClick := onCharImage;
  end;

  //Buy Window
  with Guiclass.Windows[wBuy] do
  begin
   Visible := false;
   X := halfscreenwidth - 400;
   Y := halfscreenheight - 300;
   Z := 2;
   Images[0].onClick := onPistol;
   Images[1].onClick := onRifle;
   Images[2].onClick := onEquip;

   onDestroy := CloseBuyWindow;

   imgwidth := round(Panels[0].Height/5)-18;
   //weapons
   //pistol
   for j := 0 to 4 do     //für 5 slots pro kat. durchlaufen lassen
   begin  //achtung: mit panel[0] arbeiten, die sind vertauscht...
     Guiclass.Windows[wBuy].AddImage(Panels[0].X+20,
20+Panels[0].Y+(imgwidth+10)*(j),imgwidth,imgwidth, false, 0, 0, Log.Filepath +
'data\nil.bmp');
     Guiclass.Windows[wBuy].Images[3+j].graphic := Weapons.Pistols[j].graphic;
     Images[3+j].tag := j;
     Images[3+j].onClick := onPistolItem ;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+30,
23+Panels[0].Y+(imgwidth+10)*(j), Weapons.Pistols[j].name, cWhite);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+40,
40+Panels[0].Y+(imgwidth+10)*(j), 'Damage: ' +
```

```
inttostr(round(Weapons.Pistols[j].WeaponDMG))
                            + ';      Accuracy: '+inttostr(round(Weapons.Pistols[j].Accuracy))+ ';
Cost: ' + inttostr(Weapons.Pistols[j].prize), cBlack);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddTextField(Panels[0].X+imgwidth+40,
57+Panels[0].Y+(imgwidth+10)*(j), Panels[0].Width - (imgwidth+40)-10, 100,
Weapons.Pistols[j].description, fCouriernew12);
     GUIclass.Windows[wBuy].Textfields[i].TextColor := cBlack;
     GUIclass.Windows[wBuy].Textfields[i].Space := 17;
   end;

   //rifle
   for j := 0 to 4 do     //für 5 slots pro kat. durchlaufen lassen
   begin  //achtung: mit panel[0] arbeiten, die sind vertauscht...
     AddImage(Panels[0].X+20,20+Panels[0].Y+(imgwidth+10)*(j),imgwidth,imgwidth, false, 0,
0, Log.Filepath + 'data\nil.bmp');
     Images[8+j].graphic := Weapons.Rifles[j].graphic;
     Images[8+j].tag := j;
     Images[8+j].onClick := onRifleItem;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+30,
23+Panels[0].Y+(imgwidth+10)*(j), Weapons.Rifles[j].name, cWhite);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+40,
40+Panels[0].Y+(imgwidth+10)*(j), 'Damage: ' + inttostr(round(Weapons.Rifles[j].WeaponDMG))
                            + ';      Accuracy: '+inttostr(round(Weapons.Rifles[j].Accuracy))+ ';
Cost: ' + inttostr(Weapons.Rifles[j].prize), cBlack);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddTextField(Panels[0].X+imgwidth+40,
57+Panels[0].Y+(imgwidth+10)*(j), Panels[0].Width - (imgwidth+40)-10, 100,
Weapons.Rifles[j].description, fCouriernew12);
     GUIclass.Windows[wBuy].Textfields[i].TextColor := cBlack;
     GUIclass.Windows[wBuy].Textfields[i].Space := 17;
   end;

   //equip
   for j := 0 to 4 do     //für 5 slots pro kat. durchlaufen lassen
   begin  //achtung: mit panel[0] arbeiten, die sind vertauscht...
     AddImage(Panels[0].X+20,20+Panels[0].Y+(imgwidth+10)*(j),imgwidth,imgwidth, false, 0,
0, Log.Filepath + 'data\nil.bmp');
     Images[13+j].graphic := Weapons.Equips[j].graphic;
     Images[13+j].tag := j;
     Images[13+j].onClick := onEquipItem;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+30,
23+Panels[0].Y+(imgwidth+10)*(j), Weapons.Equips[j].name, cWhite);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddText(Panels[0].X+imgwidth+40,
40+Panels[0].Y+(imgwidth+10)*(j), 'Cost: ' + inttostr(Weapons.Equips[j].prize), cBlack);
     GUIclass.Windows[wBuy].Text[i].font := fCouriernew12;
     i := GUIclass.Windows[wBuy].AddTextField(Panels[0].X+imgwidth+40,
57+Panels[0].Y+(imgwidth+10)*(j), Panels[0].Width - (imgwidth+40)-10, 100,
Weapons.Equips[j].description, fCouriernew12);
     GUIclass.Windows[wBuy].Textfields[i].TextColor := cBlack;
```

```
      GUIclass.Windows[wBuy].Textfields[i].Space := 17;
    end;
  end;
  onPistol(0,0,nil,1);
end;

procedure TGUIAdd.Render;
begin
  GUIclass.GoOrtho(Screen_Width,Screen_Height, -40, 40);
  glColor4f(1,1,1,1);
  if game.ingame then
  begin
    //Das Bild rendern.
    Renderpass1.Render;
    if Gamevariabeln.VideoConfig.Shader then
    begin
      Renderpass2.GetScreen;
      Renderpass2.Render;
      Renderpass3.GetScreen;
      glActiveTextureARB(GL_TEXTURE1);
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, Renderpass1.Texture);
      glDisable(GL_Texture_2D);
      glActiveTextureARB(GL_TEXTURE0);
      Renderpass3.Render;
    end;

    //debug
    if game.gameVar.showDebug then
    begin
      GUIclass.PrintFont(5, 5,0,5,Pchar(inttostr(FPS)+' FPS'));
      GUIclass.PrintFont(5,35,0,3,'Debug:');
      GUIclass.PrintFont(5,50,0,3,Pchar('Lookatx/y:
'+inttostr(round(Chars.CurrentChar.position.lookanglex))+'/'+inttostr(round(Chars.CurrentChar.p
osition.lookangley))));
      GUIclass.PrintFont(5,65,0,3,Pchar('x/y/z:
'+inttostr(round(Chars.CurrentChar.position.x))+'/'+floattostr(Chars.CurrentChar.position.y)+'/'+i
nttostr(round(Chars.CurrentChar.position.z))));
    end;

    if Chars.NewRespawn then
    begin
      GUIclass.PrintFont(Screen_Width div 2 - 400, 200, 1, fGhost, PChar('Respawn in ' +
          inttostr(round((Chars.NextRespawn - Eventhandler.counter)/1000)) + ' Seconds!'));
    end;

    glPushMatrix;
      //glColor4fv(@cGUIRed);
      GUIclass.PrintFont((Screen_Width div 4)  -40, Screen_Height-80, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Armor)));
      GUIclass.PrintFont((Screen_Width div 4)*2-40, Screen_Height-80, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Health)));
```

```
    GUIclass.PrintFont((Screen_Width div 4)*3-40, Screen_Height-80, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Dollar)));

    case chars.CurrentChar.currentweaponslot of
      0:  begin
            GUIclass.PrintFont(Screen_Width - 120, 20, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Rifle.magazines)));
            if Chars.CurrentChar.Rifle.Shots > 9 then
              GUIclass.PrintFont(Screen_Width - 140, 90, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Rifle.Shots)))
            else
              GUIclass.PrintFont(Screen_Width - 120, 90, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Rifle.Shots)));
          end;

      1:  begin
            GUIclass.PrintFont(Screen_Width - 120, 20, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Pistol.magazines)));
            if Chars.CurrentChar.Pistol.Shots > 9 then
              GUIclass.PrintFont(Screen_Width - 140, 90, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Pistol.Shots)))
            else
              GUIclass.PrintFont(Screen_Width - 120, 90, 0,fGhost,
PChar(inttostr(Chars.CurrentChar.Pistol.Shots)));
          end;
    end;
    //glColor4fv(@cWhite);
  glPopMatrix;

 end else
  if game.runninggame then
  begin
    GUIclass.Windows[wMain].Text[5].Visible := true;
    GUIclass.Windows[wMain].Text[6].Visible := true;
  end
  else
  begin
    GUIclass.Windows[wMain].Text[5].Visible := false;
    GUIclass.Windows[wMain].Text[6].Visible := false;
  end;

 GUIclass.Render;

 GUIclass.ExitOrtho(60,Screen_Width/Screen_Height,0.1,600);
end;

procedure TGUIAdd.Resize;
const
 _g = 15;
var
 i: integer;
begin
```

```
  with GUIclass.Windows[wMain] do
  begin
    Width := Screen_Width;
    Height := Screen_Height;
    for i := 0 to 4 do
      Text[i].X := 30;
    Text[0].Y := Screen_Height -  110;
    Text[1].Y := Screen_Height -  90;
    Text[2].Y := Screen_Height -  70;
    Text[3].Y := Screen_Height -  50;
    Text[4].Y := Screen_Height -  30;
    Text[5].Y := Screen_Height -  190;
    Text[6].Y := Screen_Height -  170;
  end;

  with GUIclass.Windows[wIngame] do
  begin
    Width := Screen_Width;
    Height := Screen_Height;
    Images[0].X := halfscreenwidth-_g;
    Images[0].Y := halfscreenheight-_g;
    Images[1].X := (Screen_Width div 4)  -100;
    Images[1].Y := Screen_Height-80;
    Images[2].X := (Screen_Width div 4)*2-100;
    Images[2].Y := Screen_Height-80;
    Images[3].X := (Screen_Width div 4)*3-100;
    Images[3].Y := Screen_Height-80;
    Images[4].X := Screen_Width - 70;
    Images[5].X := Screen_Width - 70;
  end;

  with GUIclass.Windows[wChars] do
  begin
    X := halfscreenwidth - (Width div 2);
    Y := halfscreenheight - (Height div 2);
  end;

  Console.TextField.Y := Screen_Height - 300;
end;

{
-------------------------------------------------------------------------------
        Console
-------------------------------------------------------------------------------
}
procedure ConsoleButtonClick(X,Y: integer; Element: TGUIObject; Button: integer);
begin
  if Button = 1 then
  begin
    Console.AddString(Console.Edit.Text);
    Console.ProceedInput(Console.Edit.Text);
    Console.Edit.Text := '';
```

```
    Console.Edit.MousePosition := 0;
  end;
end;

procedure OnEnter(Key: Word; Element: TGUIObject);
begin
  if Key = 13 then
    ConsoleButtonClick(0,0, Element, 1);
  Console.AddString('pushed: ' +  inttostr(Key));
end;

procedure TConsole.AddString(s: string);
begin
  ClearLastLine;
  with EditField do
  begin
    setlength(Stringlist, length(Stringlist)+1);
    Stringlist[High(Stringlist)] := '> ' + s;
    Text := Text;
  end;
  TextField.Stringlist[0] := s;
end;

procedure TConsole.AddStringAndLog(Warning, Where: string);
begin
  AddString(Where + ': ' + Warning);
  Log.AddWarning(Warning, Where);
end;

procedure TConsole.ClearLastLine;
var
  i: integer;
begin
  with EditField do
    while length(Stringlist) > 25 do
    begin
      for i := 1 to High(Stringlist) do
        Stringlist[i-1] := Stringlist[i];
      setlength(Stringlist,length(Stringlist)-1);     //needs improvement
    end;
  //for ingame textfield:
  with TextField do
  begin
    for i := High(Stringlist) downto 1 do
      Stringlist[i] := Stringlist[i-1];
    Stringlist[0] := '';
  end;
end;

constructor TConsole.Create;
begin
  inherited Create;
```

```
  Window := GUIclass.AddWindow(10,10,600,600,1,'Console', 'data\skins\black
blood\default.tga');
  with GUIclass.Windows[Window] do
  begin
    onDestroy := onCloseConsole;

    //Field
    AddEditField(20,36,560,506,'',fCouriernew12);
    EditFields[0].readonly := true;
    EditFields[0].plusx := 10;
    EditFields[0].plusy := 10;
    EditField := EditFields[0];

    //Edit
    AddEdit(20,550,420,30,'');
    Edits[0].font := fCouriernew12;
    Edits[0].TextColor := cBlack;
    Edits[0].plusx := 10;
    Edits[0].plusy := 7;
    Edit          := Edits[0];
    Edit.onKeyDown := OnEnter;

    //Button
    AddButton(460,550,100,30,'Submit');
    Buttons[0].font := fCouriernew12;
    Buttons[0].plusx := 25;
    Buttons[0].plusy := 7;
    Buttons[0].onClick := ConsoleButtonClick;
    Buttons[0].onKeyDown := OnEnter;

    Visible := true;
    Captionbar := true;
    dragevent  := true;
    font := fCouriernew13;
    MinimizeButton.Visible := true;
    MaximizeButton.Visible := true;
    CloseButton.Visible := true;
    VIsible := false;

    //Add to Child of Main Window
    GUIclass.Windows[wMain].AddChildWindow(GUIclass.Windows[Window]);
  end;
  //only ingame:
  GUIclass.Windows[wIngame].AddTextField(10, Screen_Height - 400, 300, 100, '',
fCouriernew12);
  TextField := GUIclass.Windows[wIngame].TextFields[0];
  setlength(TextField.Stringlist, 5);
  TextField.Visible := true;
  TextField.TextColor := cGUIRed;
end;

destructor TConsole.Destroy;
```

```
begin

  inherited Destroy;
end;

procedure TConsole.ProceedInput(Input: string);
begin
  Input := LowerCase(Input);
      if Input = 'butision'      then  AddString('> Lol, you are my hero!')
  else if Input = 'quit'         then  done := -1
  else if Input = 'reinit_shader' then InitShaders
  else if Input = 'exit'         then GUIclass.Windows[Window].Visible   := false
  else if Input = 'endlevel'      then game.EndLevel
  else if Input = 'showdebug'     then game.gameVar.showDebug            := true
  else if Input = 'hidedebug'     then game.gameVar.showDebug            := false
  else if Input = 'shownodes'     then ULevels.SHOWNODES                 := true
  else if Input = 'hidenodes'     then ULevels.SHOWNODES                 := false
  else if Input = 'continue'      then game.ingame                       := true
  else if Input = 'suicide'       then Chars.Char[Chars.ownChar].Health   := 0
  else if Input = 'uselists'      then USE_DISPLAY_LISTS := true
  else if Input = 'notuselists'   then USE_DISPLAY_LISTS := false
  else if Input = 'addterspawn'   then
                           begin
                             game.CurrentLevel.AddTerSpawnPoint(Chars.CurrentChar.Sphere.center)
;
                             AddStringAndLog('Added Terrorist Spawn Point: ' +
                                VectorToString(Chars.CurrentChar.Sphere.Center), 'GUIAdds');
                           end
  else if Input = 'addctspawn'    then
                           begin
                             game.CurrentLevel.AddCTSpawnPoint(Chars.CurrentChar.Sphere.center);
                             AddStringAndLog('Added CT Spawn Point: ' +
                                VectorToString(Chars.CurrentChar.Sphere.Center), 'GUIAdds');
                           end
  else if Input = '5rp2eeph3k'    then
                           begin
                             game.gameVar.isAdmin             := true;
                             AddString('You are now admin!');
                           end
  else if Input = 'reinit_weapons'then
                           begin
                             Weapons.Free;
                             Weapons := TWeapons.Create;
                           end

  //"parser"
  else if copy(Input, 0, 4) = 'grav'     then    //change gravity
                            begin
                              if isInteger(copy(Input, 6,length(Input) - 5)) then
                                game.gameVar.gravity := strtoint(copy(Input, 6,length(Input) -
5));
                              AddString('Gravity is now: ' + inttostr(game.gameVar.gravity));
```

```
                                end
  else if copy(Input, 0, 5) = 'speed'    then   //change runspeed
                                begin
                                  if isFloat(copy(Input, 7,length(Input) - 6)) then
                                    game.gameVar.speed := strtofloat(copy(Input, 7,length(Input) -
6));
                                  AddString('Speed is now: ' + floattostr(game.gameVar.speed));
                                end
  else if copy(Input, 0, 5) = 'name'     then    //change name
                                begin
                                  Chars.char[Chars.OwnChar].name := copy(Input, 6,length(Input) -
5)
                                end


  else
  if game.gameVar.isAdmin then
  begin
        if Input = 'fullmoney' then  Chars.CurrentChar.Dollar := 16000
    else if Input = 'fullhealth'then  Chars.CurrentChar.Health := 100
    else if Input = 'fullarmor' then  Chars.CurrentChar.Armor  := 100
  end;
end;

initialization
  GUIAdd := TGUIAdd.Create;

finalization
  Console.Free;
  GUIAdd.Free;

end.
```

10. MainUnit

```
unit Mainunit;


(*
* history:
* bis ca.    Engine programmiert und einen sehr grossen Teil des
* 10.5.06:    Partikelsystems fertiggestellt.
* von Mai    Literatur über Audio und Netzwerk durchgegangen,
* bis August: ausserdem die Skybox und das Orthosystem im Standart-Editor
*            von Microsoft geschrieben. Teile der Netzwerk Unit geschrieben.
*            In collision detection weitergebildet und somit eine Technik
*            entwickelt ob eine Gerade einen Kreis durchkreuzt oder eben nicht.
* 11.08.06:  Arbeit an dem Projekt wieder aufgenommen, durch einen sehr langen
*            Ausfall meines PCs war es mir unmöglich hier
*            weiterzuprogrammieren und es ging auch ein Teil der vorher
*            programmierten Sachen verloren, zum Glück konnte ich noch ein
*            ziemlich gutes Backup auftreiben.
*            zahlreiche Bugs verbessert, Skyboxen eingebaut, Orthomode eingebaut,
```

```
*           Umstieg auf Borland Delphi 2005 von Delphi 7.
* 12.08.06:  Schuss und 3ds implementiert, beides allerdings noch nicht voll funktionsfähig....
* 13.08.06:  Menuclass (inkl ocl.pas) implementiert, jedoch probleme mit dem laden...
*           ausserdem OpenAL (Open Audio Library) eingebaut, und eine
*           objektorientierte klasse von Noeska eingebaut.
*           Version 0.3 fertiggestellt. Allerdings gibt es sehr viele Bugs.
*           KI-Unit, Waffensystem eingebaut, ausserdem Schuss fertiggestellt - allerdings noch
buggy.
* 14.08.06:  KI-Grundlagen verbessert, respawn, addbots, deletebots, usw...
* 17.08.06:  Menu anzeigbar gemacht, das gleiche mit den Schriften.
*           ca. 200 zeilen überflüssiger Quellcode gelöscht
* 19.08.06:  Auf Version 0.3.5 updated.
*           Viel an den Menus rumgebastelt, Startmenu erstellt.
*           Endlich geschafft, OpenAL zum laufen zu bringen, Starttheme laufen lassen.
* 21.08.06:  NetzwerkUnits - Grundkonzept fertiggestellt, allerdings müsste das ganze getestet
*           werden,unter anderem für mehr als 2 player eine ein bisschen andere
*           Netzwerkunit geschrieben werden, da momentan nur 1 fremder Client unterstützt wird.
*           Einen rechten Teil der Mainunit dokumentiert(Deutsch)
*          Projekt durchgezählt, umfasst mittlerweile ca 3200 Zeilen ~ 60 Seiten, 19 Units
*           und 13 Units mit Code, der nicht von mir geschrieben wurde (ca 30'000 Zeilen~600
Seiten)
* 22.08.06:  3ds und msa zum laufen gebracht, nur die modells werden noch benötigt...
* 23.08.06:  An der ocl Unit gearbeitet und gemerkt, dass diese Unit so schlecht ist, dass man
*           sie eigentlich wegschmeissen sollte... Hier werden EINIGE Änderungen erforderlich sein.
* 24.08.06:  Entscheid zu einem Leveleditor, der bald in Arbeit sein wird. Dieser sollte dann auch
gerade
*           Partikeleffekte benutzen können und die jpgmap für begehbar/unbegehbar zeichnen.
* 26.08.06:  Auf Version 0.4 updated.
* 02.01.07:  Die ganze GUI wurde neu programmiert und ist nun selbst 4600 Zeilen + Editor (ca
800 Zeilen) gross.
* 28.01.07:  Momentan stehen die Levels bzw. Octrees im Vordergrund.
* 01.02.07:  Ich habe nun die Octrees mehr oder weniger fertiggestellt. 3ds muss noch besser
konvertiert werden.
* 07.02.07:  Ein Logger wurde eingearbeitet.
* 02.03.07:  Die GUI ist nun auch im Game eingebaut, mit einem neuen Skin und einigen
Fenstern.
           Nun werde ich mich wieder dem Level laden widmen.
* 17.03.07:  Kollisionen sind nun möglich, allerdings noch nicht wirklich optimal.
           Ausserdem sind Levels nun vollständig Renderbar.
* 30.03.07:  Kollisionen sind nun verbessert und die Levels sind viel besser optimiert.
* 01.04.07:  Kollisionen laufen nun immer, es gibt also keine falschen Kollisionen mehr.
* 07.04.07:  Schüsse funktionieren nun.
* 08.04.07:  Eine Konsole läuft nun.
          Durchgezählt - über 10'000 Zeilen (nur dieses Projekt, ohne alte Überreste, ohne externe
Editoren usw.
* 02.07.07:  Mittlerweile ist relativ viel kleines dazugekommen, z.B. Timebased Movement,
           ein funktionierendes Waffensystem, viele Designänderungen, viele GUI Änderungen usw.
* 13.07.07:  Ca. 13'000 Zeilen...
*-14.10.07:  Statistiken, neue Level, viele bugs ausgemerzt,
           funktionierendes Netzwerk, Schüsse etc.
           Respawn, Gravitation, Ausweichen, wenn man an der Wand läuft etc.
           Eigentlich ist nun ein Punkt erreicht wo nur noch Kleinigkeiten gemacht werden müssen.
```

```
*)

(*




*)

interface

uses
  //System - Uses
  Windows,
  sysutils,
  SDL,
  DGLopenGL,
  dialogs,
  gl3ds,
  textures,
  GLFrustum,
  //Programm - Uses
  UNIT_SDL,
  MovementUnit,
  CamUnit,
  GameVARUnit{überarbeiten},
  EventHandlerUnit,
  other_unit,
  PartikelUnit,
  oooal,
  UKI,
  GUI,
  GUIAdds,
  Basics,
  ULevels,
  ULogger,
  UCharacters,
  UNetwork,
  UShader;

type

  TMusic = record
    Aerzte_theme:     integer;
  end;

  Tgame = class(Tobject)
  private
    _ingame:          boolean;              //ob man ingame ist, oder eben irgendwo in den Menus.
    _showCursor:       boolean;
```

```
    procedure setingame(Value: boolean);
    procedure setShowCursor(Value: boolean);
  public
    CurrentLevel:     TLevel;
    map:              Tmap;
    Music:            TMusic;
    gameVar:          TgameVar;

    //debugging
    debugvec:         TVector3d;

    //ortho
    resttime:         integer;              //in Sekunden

    LoadedMenus:      boolean;
    runninggame:      boolean;

    procedure         start;
    procedure         Draw_Game;
    procedure         EndLevel;
    procedure         Render;
    procedure         startlevel(name: string);
    procedure         initlevel(name: string);
    procedure         onError(ErrorMsg, Where: string);

    property ingame: boolean read _ingame write setingame;
    property ShowCursor: boolean read _showcursor write setShowcursor;
  end;

var
  game:               Tgame;
  tools:              Ttools;
  Movement:           TMovement;
  gameVariabeln:      TgameVariabeln;
  SDL_Unit:           TSDL_Unit;
  Eventhandler:       TEventHandler;

  Screen_width:       integer   = 1024;
  Screen_height:      integer   = 800;
  halfscreenwidth:    integer   = 400;
  halfscreenheight:   integer   = 300;

  pobject, fobject, vobject: integer;
  outline: cardinal;

  gebloedel: integer;

implementation

//die Hauptprozedur, aus dieser geht das programm nie raus, alle inits sind hier drin und auch die
meisten bugs ;)
//warscheinlich auch die einzige prozedur in dieser unit!
```

```
//timer manage:       //kann man eigentlich streichen, unwichtig...
//1. FPS
//2. Movement
//3. Partikel

procedure Tgame.Initlevel(name: string);
var
  fname: string;
begin
  Log.AddStatus('Loading new Level: ' + name, 'Main');
  //if the Level is not converted to 3ds then that converts it.
  fname := 'data\maps\'+name + '\map';
  if not FileExists(fname + '.lvl') then
  begin
    if FileExists(fname + '.3ds') then
      Convert3dstoLvl(name,500);
  end;

  //Load Level
  CurrentLevel := TLevel.Create(name);

  Chars := TCharacters.Create;
  KI:= TKI.create;   //init KI
  cam.Init(chars.CurrentChar);   //currentplayer must be defined first otherwise -> exception

  ingame := true;

  //Charauswahlbildschirm anzeigen
  GUIclass.Windows[wChars].Visible := true;
  game.ShowCursor := true;

  // -------------------- vorübergehendes zeug "hack" ----------------------

  //game.weapons.currentweaponslot:=KI.addbot(2,0,2);
  SDL_WarpMouse(round(Screen_Width/2),round(Screen_Height/2));

  resttime := 300;                    //die Zeit, bis ein Levelwechsel erfolgt.
  Movement.start(0,0);

  //game.camera.angley:= Pi/2;            //um die kamera schön einzustellen
  //Chars.AddChar(0,0,0,0);

  partikel.addpartikel(1,10,1,10,     1,1,1,    1,1,1,0);
  partikel.addpartikel(1,1,4,5,       5,5,5,    1,0,0,90);
  //partikel.addpartikel(1,0,0,0,       1,1,1,    1,1,1,0);
  partikel.addpartikel(1,2,1,0,       1,1,1,    1,1,1,0);
  partikel.addpartikel(1,2,1,15,      10,10,10,    1,1,1,0);

  Partikel.deletepartikel(0);

  {partikel.addpartikel(1,3,0,0,       1,1,1,    1,1,1,0);
  partikel.addpartikel(1,4,0,0,       1,1,1,    1,1,1,0);
```

```
  partikel.addpartikel(1,5,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,6,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,7,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,8,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,9,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,10,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,11,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,12,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,13,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,14,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,15,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,16,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,17,0,0,      1,1,1,    1,1,1,0);
  partikel.addpartikel(1,18,0,0,      1,1,1,    1,1,1,0);
  }

  Log.AddStatus('3..2..1..Go! Level: "' + name +'" has been loaded', 'UMain');
end;

//zuerst partikeleffekte, wichtig, wegen depthtest   ?!?!?! sicher?!
procedure Tgame.Draw_Game;
var
  i,j: integer;
begin
  if ingame then
  begin
    glMatrixMode(GL_TEXTURE);
    inc(gebloedel);
    //glTranslatef(gebloedel/50,gebloedel/50,gebloedel/50);
    glMatrixMode(GL_MODELVIEW);

    Frustum.Calculate;   //Berechnet, wie die Kamera ausgerichtet ist.

    Chars.RenderChars;

    glLoadIdentity;
    CurrentLevel.Render;
    //glLoadIdentity;

    // Zeichne Dreieck
    glcolor4f(1,0,0,1);
    glBegin( GL_TRIANGLES );
      glVertex3f( 0.0, 2.0, 0.0 );
      glVertex3f( 1.0, 0, 0.0 );
      glVertex3f( -1.0, 0, 0.0 );
    glEnd;

    //Partikel:
    partikel.PartikelRender;

    glMatrixMode(GL_TEXTURE);
    glloadidentity;
```

```
    glMatrixMode(GL_MODELVIEW);

    Renderpass1.GetScreen;
  end;

  //GUI:
  GUIAdd.Render;     //Ruft die ganzen Befehle auf um die GUI zu Rendern.
end;

procedure Tgame.EndLevel;
var
  i: integer;
begin
  runninggame     := false;
  ingame          := false;
  Net.EndLevel;
  for i := 0 to partikel.partikelnr - 1 do
    partikel.running[i] := false;
  CurrentLevel.Free;
  KI.Free;
  Chars.Free;
  CurrentLevel    := nil;
  KI              := nil;
  Chars           := nil;
end;

procedure Tgame.onError(ErrorMsg, Where: string);   //nur gebrauchen während das programm
läuft (not initialization or finalization)
begin
  Log.AddError(ErrorMsg, Where);
  with GUIclass.Windows[wNetError] do      //eigentlich wError, aber früher wurde die Prozedur nur
im Netzwerk gebraucht.
  begin
    Text[0].Text := ErrorMsg + ' - Look at .log file!';
    Width := Text[0].Width + 40;
    Height := 100;
    X := halfscreenwidth - (Width div 2);
    Y := halfscreenheight - 50;
    Visible := true;
    Caption := 'Error in ' + where;
  end;
  Console.AddString(ErrorMsg + ' - ' + where);
  EndLevel;
end;

procedure Tgame.Render;
begin
  SDL_UNIT.glHandleEvents;                          //Maus & Keyboard - Events werden
durchgegangen
  EventHandler.TimebasedMovement;                       //Ersatz für die ganzen Timer...
  Net.Update;                                  //Geht das Netzwerk durch.
  if not game.ShowCursor then
```

```
    SDL_WarpMouse(halfscreenwidth,halfscreenheight);          //Die Maus wird in die Mitte des
Bildschirms gesetzt

  glClear( GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT );       //Screen- und Tiefenbuffer
bereinigen
  glLoadIdentity;                                          //Die Null-Matrizen werden geladen

  if ingame then
    cam.setcam(Chars.CurrentChar);                            //Die Kamera wird auf den richtigen Ort
gesetzt

  Draw_Game;                                               //alles was gezeichnet wird - player, map usw

  SDL_GL_SwapBuffers;                                      //Buffer-Wechseln ==> Anzeigen

  Inc(FPSCount);                                           //Dafür da, um die Frames zu zählen
  sleep(game.gameVAR.sleep);                                  //Sleep, um den Prozessor nicht 100%
auszulasten
end;

procedure Tgame.setingame(Value: boolean);
begin
  _ingame := Value;
  if Value then
  begin
    GUIclass.Windows[1].Visible := true;
    GUIclass.Windows[0].Visible := false;
    ShowCursor := false;   // Cursor ausblenden
  end
  else
  begin
    GUIclass.Windows[0].Visible := true;
    GUIclass.Windows[1].Visible := false;
    ShowCursor := true;   // Cursor einblenden
  end;
end;

procedure Tgame.setShowCursor(Value: boolean);
begin
  if Value then
    SDL_ShowCursor(1)    // Cursor einblenden
  else
    SDL_ShowCursor(0);   // Cursor ausblenden
  _showcursor := value;
end;

procedure Tgame.start;
begin
  Decimalseparator := '.';        //wichtig wegen 3ds, aus textdateien zu laden...

  //Variabeln richtig setzen...
  LoadedMenus := false;
```

```
  runninggame := false;
  _ingame := false;
  _showcursor := true;

  //Die eigenen Dienste initialisieren und Speicher freigeben.
  SDL_Unit:=      TSDL_Unit.Create;
  Movement:=      TMovement.create;
  EventHandler:=  TEventHandler.Create;
  tools:=         TTools.Create;
  OpenAL:=        TOpenalclass.Create;

  gameVariabeln:= TgameVariabeln.Create(ExtractFilePath(ParamStr(0))+'config.ini'); //Variabeln
richtig setzen

  gameVariabeln.setdefault;
  gameVariabeln.LoadFromIni;

  //dienste initialisieren
  SDL_Unit.Init;                  //SDL & OpenGL starten

  //Audio Einstiegssong abspielen
  game.Music.aerzte_theme := openal.addsound('data\sound\aerzte_theme.wav');
  OpenAL.sounds[game.Music.Aerzte_theme].Play;

  //eher unwichtige einstellungen um aus den Variabeln keine bugs zu erhalten
  halfscreenwidth:= Screen_width div 2;      //halfscreenwidth setzen um Leistung zu sparen, nicht
immer Screen_width/2 zu rechnen
  halfscreenheight:= Screen_height div 2;    //...

  Weapons := TWeapons.Create;               //Alle Waffen laden.
  GUIadd.Init;                            //Initialisiert die GUI.

  Partikel.Init;
  tools.Loadskybox;

  Log.AddStatus('The Initialization has been done', 'Main');

  SDL_UNIT.glHandleEvents;                  //Die Mouse & Keyboard Events werden noch einmal
durchgegangen, bevor man dann mit dem Mainloop beginnt.

  LoadedMenus := true;

  InitShaders;
  //Tools.Fog(true);

  // Eintritt in Main-Loop
  while ( Done <> -1 ) do
  begin
    Render;
  end;

  //glDeleteObjectARB(pObject);
```

```
  Log.AddStatus('Deinitialization has been started', 'Main');

  gameVariabeln.SaveToIni;

  EndLevel;

  //Zerstört die ganzen Klassen.
  Eventhandler.Free;
  Movement.Free;
  gameVariabeln.Free;
  tools.Free;
  OpenAL.Free;

  SDL_Unit.Quit_App;
  SDL_Unit.Free;
end;

procedure Tgame.startlevel(name: string);
var
  fname: string;
begin
  if runninggame then
  begin
    GUIclass.Windows[wChangeLevel].Visible := true;
    tempLevelName := name;
  end else
  begin
    runninggame := true;
    Initlevel(name);
  end;
end;

end.
```

11. MovementUnit

```
unit MovementUnit;

interface

uses
  Windows,
  SDL,
  Basics,
  ULevels,
  UOctree,
  UCharacters,
  sysutils;

type
  TPolyList = array of PPolygon;
```

```
  TMovement = class
    TimerID:  PSDL_TimerID;
    oldtimer: boolean;
    CurrentYSpeed: single;
    constructor Create;
    destructor Destroy; override;
    function MovementTimer( interval : UInt32; param : Pointer ) : UInt32;
    procedure Start(X,Z: integer);
    function HasCollision(const Node: TNode; const Char: TCharacter; var ColPolys: TPolyList):
boolean;
  end;


implementation

uses
  Mainunit, Unit_SDL, GUIAdds;


{-------------------------------------------------------------------------------}
{                    Timer für die Bewegungen                                   }
{-------------------------------------------------------------------------------}
constructor TMovement.Create;
begin
  inherited Create;
  OldTimer := false;                //Eigenschaft, ob der alte Movement timer entfernt werden muss
oder nicht.
end;

destructor TMovement.Destroy;
begin
  if oldtimer then SDL_RemoveTimer(timerid);
  inherited Destroy;
end;

//Diese Funktion ist nur für den CurrentChar.
//Achtung ist relativ komplex geschrieben, enthält aber sehr interessanten Code.
function TMovement.MovementTimer( interval : UInt32; param : Pointer ) : UInt32;
var
  counter:  single;
  oldposx:  single;
  oldposy:  single;
  oldposz:  single;
  i,j:      integer;
  CurPoly:  PPolygon;
  ColPolys, ColPolys2: TPolyList;
  pos, normal, point, tempv1, tempv2, newpos:      TVector3f;
  newvelocity:  single;
  hascol:   boolean;
begin
  with Movement do  //wichtig um Exceptions zu vermeiden, weil SDL spinnt.
  begin
```

```
    if not game.ShowCursor then    //Wenn der Cursor angezeigt wird dann ist man in der GUI und
die Figur soll sich nicht bewegen.
      if Chars.CurrentChar.Index = Chars.OwnChar then
      begin
        if game.gameVar.keys.backw.state or game.gameVar.keys.forw.state or
           game.gameVar.keys.left.state or game.gameVar.keys.right.state then
        begin
          counter := 0;

          if game.gameVar.keys.left.state then
          begin
            if game.gameVar.keys.forw.state then
              counter := counter + 0.25;
            if game.gameVar.keys.backw.state then
              counter := counter - 1.25;
            counter := counter + 1.5;
          end;

          if game.gameVar.keys.right.state then
          begin
            if game.gameVar.keys.forw.state then
              counter := counter - 0.25;
            if game.gameVar.keys.backw.state then
              counter := counter - 0.75;
            counter := counter + 0.5;
          end;

          if game.gameVar.keys.backw.state then
            counter := counter + 1;


          with Chars.CurrentChar do
          begin
            oldposx:=position.x;
            oldposy:=position.y;
            oldposz:=position.z;

            Chars.CurrentChar.PlusMove(cos(DEGTORAD*position.lookanglex +
counter*Pi)*game.gameVar.speed,
                                       0,
                                       sin(DEGTORAD*position.lookanglex +
counter*Pi)*game.gameVar.speed);

            setlength(ColPolys, 0);
            if HasCollision(game.CurrentLevel.Octree.mainnode, Chars.CurrentChar, ColPolys) then
            begin
              //nächstes Polygon berechnen   //müsste noch gemacht werden...
              j := 0;
              //for i := 1 to Length(ColPolys) - 1 do
                //if  then
                  //j := i;
```

```
            pos    := Vec3f(Sphere.center.x, Sphere.center.y, Sphere.center.z);
            normal := ColPolys[j].Vertexes[0].normal;
            //punkt berechnen, der von der normale durch den Kugelpunkt und der Ebene gegeben
ist.
            point := minusvector(pos, malvector(normal, skalar(normal, pos) - ColPolys[j].d));

            tempv1 := minusvector(plusvector(point, normal), pos);       //+ normale
            tempv2 := minusvector(minusvector(point, normal), pos);      //-normale
            if skalar(tempv1, tempv1) < skalar(tempv2, tempv2) then      //prüfen welche seite...
            begin
              newpos := plusvector(point, malvector(normal, Sphere.radius*1.001));
            end else
            begin
              newpos := plusvector(point, malvector(normal, -Sphere.radius*1.001));
            end;

            MoveSphere(newpos);    //Bewege den Char an die richtige Stelle...

            //erneut überprüfen ob eine Kollision stattfindet...
            setlength(ColPolys, 0);
            if HasCollision(game.CurrentLevel.Octree.mainnode, Chars.CurrentChar, ColPolys) then
              Move(oldposx, oldposy, oldposz);
          end;
        end;
      end;
      with Chars.CurrentChar do
      begin
        //Gravitation berechnen        //gravity wird mit int angegeben. und ist standartmässig 98.
        newvelocity := CurrentYSpeed - game.gameVar.gravity*0.01;   //
game.gameVar.gravity/10*0.02*5 -> konstante, weil die welt kleiner ist.
        oldposy := position.y;
        //neue Position setzen, nach der Gravitation
        Move(position.x, oldposy + 0.02*Newvelocity, position.z);
        setlength(ColPolys, 0);
        if HasCollision(game.CurrentLevel.Octree.mainnode, Chars.CurrentChar, ColPolys) then
        begin
          i := -1;
          for i := 0 to length(ColPolys) - 1 do
            // bei einer Neigung von > 0.5 rutscht der charakter weg.
            if ColPolys[i].Vertexes[0].normal.y > 0.7 then
              j := i
            else
              hasCol := true;
          if hasCol then
          begin
            CalcDropDMG(CurrentYSpeed);
            CurrentYSpeed := 0;      //wenn eine Kollision stattfindet, dann ist der speed = 0.
            Move(position.x, oldposy, position.z)
          end
          else
            if i >= 0 then
            begin
```

```
            pos    := Vec3f(Sphere.center.x, Sphere.center.y, Sphere.center.z);
            normal := ColPolys[j].Vertexes[0].normal;
            //punkt berechnen, der von der normale durch den Kugelpunkt und der Ebene gegeben
ist.
            point := minusvector(pos, malvector(normal, skalar(normal, pos) - ColPolys[j].d));

            tempv1 := minusvector(plusvector(point, normal), pos);       //+ normale
            tempv2 := minusvector(minusvector(point, normal), pos);      //-normale
            if skalar(tempv1, tempv1) < skalar(tempv2, tempv2) then      //prüfen welche seite...
            begin
              newpos := plusvector(point, malvector(normal, Sphere.radius));
            end else
            begin
              newpos := plusvector(point, malvector(normal, -Sphere.radius));
            end;

            MoveSphere(newpos);

            if HasCollision(game.CurrentLevel.Octree.mainnode, Chars.CurrentChar, ColPolys) then
              Move(pos.x, oldposy, pos.z);   //wichtig: pos! nicht position.

            //die geschwindigkeit wird durch den sinus gedrosselt, bei einer "schrägen" Ebene.
            CurrentYSpeed := sin(ColPolys[i].Vertexes[0].normal.y)*NewVelocity;
          end;
        end else
          CurrentYSpeed := NewVelocity;
        if position.y < -100 then dead := true;
      end;
    end;
  end;
  Result := 20;
end;

procedure TMovement.Start(X,Z: integer);
var
  i: integer;
begin
  if oldtimer then SDL_RemoveTimer(timerid);
  timerid := SDL_AddTimer(20, @TMovement.MovementTimer, nil);
  oldtimer:= true;
  for i:=0 to High(Chars.char) do
    with Chars.char[i] do
    begin
      position.x:= X;
      position.y:= 0;
      position.z:= Z;
      health    := 100;
      armor     := 0;
    end;
  Chars.CurrentChar.Sphere.Center.x := Chars.CurrentChar.position.x;
  //game.CurrentChar.Sphere.Center.y := 1; //temp
  Chars.CurrentChar.Sphere.Center.z := Chars.CurrentChar.position.z;
```

```
  //game.CurrentChar.Sphere.radius   := 0.5;
end;

function TMovement.HasCollision(const Node: TNode; const Char: TCharacter; var ColPolys:
TPolyList): boolean;  //true -> there is a collision
var
  i: integer;
begin
  result := false;
  with Node do
  begin
    if SphereInNode(Char.Sphere) then
    begin
      //Check Triangles
      for i := 0 to numv-1 do
      begin
        if Col_Triangle_Sphere(poly[i],Char.Sphere) then
        begin
         { if Col_Triangle_Sphere(poly[i],Char.Legsphere) or
             Col_Triangle_Sphere(poly[i],Char.Headsphere) or
             Col_Triangle_Sphere(poly[i],Char.Chestsphere) then
          begin                }
            result := true;
            poly[i].material := 0;   //DELETE

            setlength(ColPolys, Length(ColPolys) + 1);
            ColPolys[High(ColPolys)] := poly[i];
            //Exit;
          //end;
        end;
      end;

      //Check Child-Nodes
      for i := 0 to numc-1 do
        if HasCollision(Children[i], Char, ColPolys) then   //geht rekursiv durch
        begin
          result := true;
          //Exit;
        end;
    end;
  end;
end;

end.
```

12. oooal

```
unit oooal;

interface
```

```
uses al, altypes, alut, sysutils, classes;

type
	TAlBuffer = class
	private
	  _buffer: TALuint;
	  _format: TALenum;
	  _size: TAlSizei;
	  _initsamplerate: TALSizei;
	  _loop: TALInt;
	protected
	public
	  constructor Create; //create sound buffer
	  destructor Destroy; override; //destroy sound buffer
	  property buffer: TALuInt read _buffer; //get the buffer id
	  property format: TAlEnum read _format write _format; //the format of the buffer
	  property size: TAlSizei read _size write _size; //the size of the buffer
	  property initsamplerate: TALSizei read _initsamplerate write _initsamplerate; //sets the
initial sample rate (used by set newsamplerate)
	  property loop: TALInt read _loop write _loop; //is the sound object looped?
	  procedure LoadFromFile(filename: string); //loads sample data from a wav file
	  procedure LoadFromStream(stream: tmemorystream); //loads sample data from a stream
	  procedure LoadFromPointer(data: pointer); //loads sample data from a pointer
	end;

	TAlSource = class
	private
	  _source: TALuint;
	protected
	public
	  constructor Create; //create sound source
	  destructor Destroy; override; //destroy sound source
	  property source: TALuInt read _source; //get the source id
	  procedure AssignBuffer(value: TALuint); //assign a buffer to a source
	end;

	TAlObject = class
	private
	  _pos: array [0..2] of TALFloat;
	  _vel: array [0..2] of TALFloat;
	  _buffer: TAlBuffer;
	  _source: TAlSource;
	  _pitch: TALFloat;
	  _samplerate: TALSizei;
	  _gain: TALFloat;
	  _playing: boolean;
	  _name: string;
	  _id: TAlInt;
	  procedure SetNewSampleRate(Value: TAlSizei);
	protected
	public
	  constructor Create; //create sound object
```

```
        destructor Destroy; override; //destroy sound object
        procedure Update; //passes changes in object to openal
        procedure Play; //tells openal to play the sound object
        procedure Pause; //tells openal to pause the sound object
        procedure Stop;  //tells openal to stop the sound object
        property xpos: TALFloat read _pos[0] write _pos[0]; //position in 3d space
        property ypos: TALFloat read _pos[1] write _pos[1]; //position in 3d space
        property zpos: TALFloat read _pos[2] write _pos[2]; //position in 3d space
        property xvel: TALFloat read _vel[0] write _vel[0]; //movement
        property yvel: TALFloat read _vel[1] write _vel[1]; //movement
        property zvel: TALFloat read _vel[2] write _vel[2]; //movement
        property pitch: TALFloat read _pitch write _pitch; //make the sound object play higher or
lower
        property samplerate: TALSizei read _samplerate write SetNewSampleRate; //sets a new
sample rate (does so by changing pitch)
        property gain: TALFloat read _gain write _gain; //the volume at what the sound object is
played
        property playing: boolean read _playing write _playing; //is the sound object currently
playing?
        property name: string read _name write _name; //the name of the sound object
        property id: TALInt read _id write _id; //an id for the sound object?
        property buffer: TAlBuffer read _buffer write _buffer;
        property source: TAlSource read _source write _source;
      end;

      TOpenALclass = class(Tobject)
        sounds:      array of TAlObject;
        numsounds:   integer;
        constructor Create;
        destructor Destroy; override;
        function  addsound(path: string): integer;
        procedure deletesound(nr: integer);
      end;

var
  OpenAL:           TOpenALclass;

implementation

uses Mainunit;

constructor Talbuffer.Create;
begin
  alGetError; //clear any previous error
  // create a buffer
  AlGenBuffers(1, @_buffer);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot create Buffer');
end;

destructor TalBuffer.Destroy;
begin
  alGetError; //clear error
```

```
  //delete the buffer
  AlDeleteBuffers(1, @_buffer);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot delete Buffer');
end;

procedure TalBuffer.LoadFromFile(filename: string);
var
  data: TALVoid;

begin
  alGetError; //clear any previous error
  //load the wavedata from the file
  AlutLoadWavFile(filename, _format, data, _size, _initsamplerate, _loop);
  //assign the wavedata to the buffer
  AlBufferData(_buffer, _format, data, _size, _initsamplerate);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot assign wave data to
buffer');
  //remove the wavedata from memory
  AlutUnloadWav(_format, data, _size, _initsamplerate);
end;

procedure TalBuffer.LoadFromStream(stream: tmemorystream);
var
  data: TALVoid;

begin
  alGetError; //clear any previous error
  //load the wavedata from the file
  AlutLoadWAVMemory(@stream, _format, data, _size, _initsamplerate, _loop);
  //assign the wavedata to the buffer
  AlBufferData(_buffer, _format, data, _size, _initsamplerate);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot assign wave data to
buffer');
  //remove the wavedata from memory
  AlutUnloadWav(_format, data, _size, _initsamplerate);
end;

procedure TalBuffer.LoadFromPointer(data: pointer);
begin
  alGetError; //clear any previous error
  //assign the wavedata to the buffer
  AlBufferData(_buffer, _format, data, _size, _initsamplerate);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot assign wave data to
buffer');
end;

constructor TalSource.Create;
begin
  alGetError; //clear any previous error
  //create a source
  AlGenSources(1, @_source);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot create Source');
```

```
end;

destructor TalSource.Destroy;
begin
  alGetError; //clear any previous error
  //delete the source
  AlDeleteSources(1, @_source);
  if alGetError <> AL_NO_ERROR then raise Exception.Create('Cannot delete Source');
end;

procedure TalSource.AssignBuffer(value: TALuint);
begin
  //assign the buffer to the source
  AlSourcei ( _source, AL_BUFFER, value);
end;

constructor TalObject.Create;
begin

  _buffer:=TAlBuffer.Create;
  _source:=TALSource.Create;

  // set default values
  gain := 1.0;
  pitch:= 1.0;
  xpos := 0.0;
  ypos := 0.0;
  zpos := 0.0;
  xvel := 0.0;
  yvel := 0.0;
  zvel := 0.0;
  _buffer.loop := AL_TRUE;

  playing := false;
  _buffer.initsamplerate:=44800;

end;

destructor TalObject.Destroy;
begin
  //for the lazy among us
  if _playing then stop;

  alSourceUnqueueBuffers(_source.source, 1, @_buffer.buffer);

  //deassign the buffer from the source (solves a potential memory leak)
  AlSourcei ( _source.source, AL_BUFFER, 0);

  if _buffer <> nil then _buffer.Free;
  if _source <> nil then _source.Free;

  inherited destroy;
```

```
end;

Procedure TalObject.SetNewSamplerate(Value: TAlSizeI);
begin
_samplerate:=Value;
_pitch:=Value / _buffer.initsamplerate;
end;

Procedure TalObject.Update;
begin
 //pass the 'changed' values on to openal
 AlSourcef ( _source.source, AL_PITCH, _pitch );
 AlSourcef ( _source.source, AL_GAIN, _gain );
 AlSourcefv ( _source.source, AL_POSITION, @_pos);
 AlSourcefv ( _source.source, AL_VELOCITY, @_vel);
 AlSourcei ( _source.source, AL_LOOPING, _buffer.loop);
end;

Procedure TalObject.Play;
begin
  AlSourcePlay(_source.source);
  playing:=true;
end;

Procedure TalObject.Stop;
begin
  playing:=false;
  AlSourceStop(_source.source);
end;

Procedure TalObject.Pause;
begin
  AlSourcePause(_source.source);
end;

constructor TOpenALclass.Create;
begin
  numsounds:=0;
  AlutInit;
end;

destructor TOpenALclass.Destroy;
var
  i: integer;
begin
  {for i := 0 to Length(Sounds) - 1 do
  begin
    if sounds[i].playing then
      sounds[i].Stop;
    sounds[i].Destroy;
  end;     }
end;
```

```
function  TOpenALclass.addsound(path: string): integer;
begin
  inc(numsounds);
  setlength(sounds,numsounds+2);
  sounds[numsounds]:=TAlobject.create;
  sounds[numsounds].Create;
  OpenAL.sounds[numsounds].buffer.LoadFromFile(path);
  OpenAL.sounds[numsounds].source.AssignBuffer(OpenAL.sounds[numsounds].buffer.buffer);
  sounds[numsounds].Update;
  result:=numsounds;
end;

procedure TOpenALclass.deletesound(nr: integer);
begin
  sounds[nr].Stop;
  sounds[nr].Destroy;
end;

end.
```

13. PartikelUnit

```
unit PartikelUnit;

interface

uses
  pfxImp, pfxCore, IniFiles, SDL, textures, windows, dglopengl;

type
  TPartikel = class(Tobject)
    partikelnr:         integer;
    Partsettings:       array of TFXSettings;
    Effects:            array of TExampleFX;
    running:            array of boolean;
    PPartTimer:         PSDL_TimerID;
    procedure Init;
    function  addpartikel(which: integer; translatex,translatey,translatez: double;
              scalex,scaley,scalez: double; rotatex,rotatey,rotatez, angle: double): integer;      //
ÜBERARBEITEN
    function  AdvanceTimer( interval : UInt32; param : Pointer ) : UInt32;
    procedure PartikelRender;
    procedure deletepartikel(which: integer);
  end;

var
  Partikel:           TPartikel;

implementation
```

```
//init, wird am anfang aufgerufen um die partikel zu laden
procedure TPartikel.Init;
var
  ini:  Tinifile;
  i:    integer;
begin
  partikelnr:=0;              //anzahl der verschiedenen Partikelanimationen
  setlength(Partsettings, 2);   //anzahl der Effekte, also ini files
  setlength(Effects, 2);
  setlength(running, 1001);     //laufende effekte
  for i := 0 to 1002 do
    running[i]:= false;
  Partsettings[0].Path:= 'C:\Dokumente und Einstellungen\David\Eigene
Dateien\opengl\matura\bigtest1\bluefire.ini';
  Partsettings[1].Path:= 'C:\Dokumente und Einstellungen\David\Eigene
Dateien\opengl\matura\bigtest1\winampeffekt.ini';

  //liest jetzt die einzelnen ini files aus...
  for i:=0 to 1 do
  begin
    ini:= TInifile.Create(Partsettings[i].path);
    with PartSettings[i] do
    begin
      particlenr           :=   ini.ReadInteger('Effect.FXSettings', 'particlenr',           600);
      START_LIVESPAN       :=   ini.ReadInteger('Effect.FXSettings', 'START_LIVESPAN',
1000);
      EMISSION_RATE        :=   ini.ReadInteger('Effect.FXSettings', 'EMISSION_RATE',         2);
      PARTICLES_PER_EMISSION :=   ini.ReadInteger('Effect.FXSettings',
'PARTICLES_PER_EMISSION',  1);
      TexPath              :=   ini.ReadString( 'Effect.FXSettings', 'TexPath',              'Fire.tga');
      agedurch             :=   ini.ReadInteger('Effect.FXSettings', 'agedurch',             100);
      ifage                :=   ini.ReadInteger('Effect.FXSettings', 'ifage',                100);
      randomr              :=   ini.Readbool(   'Effect.FXSettings', 'randomr',              false);
      randomg              :=   ini.Readbool(   'Effect.FXSettings', 'randomg',              true);
      randomb              :=   ini.Readbool(   'Effect.FXSettings', 'randomb',              true);
      colr                 :=   ini.ReadFloat(  'Effect.FXSettings', 'colr',                 1);
      colg                 :=   ini.ReadFloat(  'Effect.FXSettings', 'colg',                 0.7);
      colb                 :=   ini.ReadFloat(  'Effect.FXSettings', 'colb',                 0.25);
      randomVelx           :=   ini.ReadInteger('Effect.FXSettings', 'randomVelx ',          4);
      randomVely           :=   ini.ReadInteger('Effect.FXSettings', 'randomVely ',          6);
      randomVelz           :=   ini.ReadInteger('Effect.FXSettings', 'randomVelz ',          1);
      randomPosx           :=   ini.ReadInteger('Effect.FXSettings', 'randomposx',           10);
      randomPosy           :=   ini.ReadInteger('Effect.FXSettings', 'randomposy',           5);
      randomPosz           :=   ini.ReadInteger('Effect.FXSettings', 'randomposz',           1);
      startrandomVelx      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomVelx',      60);
      startrandomVely      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomVely',      80);
      startrandomVelz      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomVelz',      40);
      startrandomPosx      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomposx',      20);
      startrandomPosy      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomposy',      5);
      startrandomPosz      :=   ini.ReadInteger('Effect.FXSettings', 'startrandomposz',      5);
      durchVelx            :=   ini.ReadFloat(  'Effect.FXSettings', 'durchVelx',            100);
      durchVely            :=   ini.ReadFloat(  'Effect.FXSettings', 'durchVely',            100);
```

```
    durchVelz          :=  ini.ReadFloat(  'Effect.FXSettings', 'durchVelz',          1);
    durchPosx          :=  ini.ReadFloat(  'Effect.FXSettings', 'durchPosx',          100);
    durchPosy          :=  ini.ReadFloat(  'Effect.FXSettings', 'durchPosy',          100);
    durchPosz          :=  ini.ReadFloat(  'Effect.FXSettings', 'durchPosz',          1);
    startdurchVelx     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchVelx',     100);
    startdurchVely     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchVely',     100);
    startdurchVelz     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchVelz',     100);
    startdurchPosx     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchPosx',     100);
    startdurchPosy     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchPosy',     100);
    startdurchPosz     :=  ini.ReadFloat(  'Effect.FXSettings', 'startdurchPosz',     100);
    plusVelx           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusVelx',           -0.02);
    plusVely           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusVely',           0.03);
    plusVelz           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusVelz',           -1);
    plusPosx           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusPosx',           -0.05);
    plusPosy           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusPosy',           0);
    plusPosz           :=  ini.ReadFloat(  'Effect.FXSettings', 'plusPosz',           -1);
    startplusVelx      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusVelx',      -0.3);
    startplusVely      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusVely',      0.2);
    startplusVelz      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusVelz',      -0.2);
    startplusPosx      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusPosx',      -0.1);
    startplusPosy      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusPosy',      0);
    startplusPosz      :=  ini.ReadFloat(  'Effect.FXSettings', 'startplusPosz',      0);
   end;
  end;
end;

//um einen effekt zu adden, während der laufzeit des programmes...
//läuft nicht so wie sie die procedure sollte  - ÜBERARBEITEN
function TPartikel.addpartikel(which: integer; translatex,translatey,translatez: double;
        scalex,scaley,scalez: double; rotatex,rotatey,rotatez, angle: double): integer;
var
  i, j: integer;
  frei: boolean;
begin
  frei:= false;
  inc(partikelnr);
  for i:=0 to partikelnr - 1 do          //geht durch die partikel durch um zu schauen ob memory
frei ist
    if not running[i] then
    begin
      frei := true;
      j    := i;
    end;
  if not frei then
  begin
    setlength(Effects, partikelnr + 1);
    j := partikelnr;
  end else dec(partikelnr);

  Effects[j] := TExampleFX.Create;
  Effects[j].FXSettings := PartSettings[which];
  LoadTexture(Effects[j].FXSettings.Texpath, Effects[j].FireTex, false);
```

```
  Effects[j].Container := TPfxContainer.Create(Effects[j].FXSettings.particlenr);
  Effects[j].Advance(25);
  running[j] := true;
  Effects[j].translate.X := translatex;
  Effects[j].translate.y := translatey;
  Effects[j].translate.z := translatez;

  Effects[j].rotate.X := rotatex;
  Effects[j].rotate.y := rotatey;
  Effects[j].rotate.z := rotatez;
  Effects[j].rotate.angle := angle;

  Effects[j].scale.X := scalex;
  Effects[j].scale.y := scaley;
  Effects[j].scale.z := scalez;

  //Effects[j].FXSettings := PartSettings[which];       //setzt den richtigen Effekt, zb blue fire; mit
which kann angegeben werden, welcher das sein soll.
  SDL_RemoveTimer(PPartTimer);
  PPartTimer := SDL_AddTimer(25, @TPartikel.AdvanceTimer, nil);
  result:=j;
end;

function TPartikel.AdvanceTimer( interval : UInt32; param : Pointer ) : UInt32;
var
  i: integer;
begin;
  for i := 0 to Partikel.partikelnr do
    if Partikel.running[i] then
      Partikel.Effects[i].Advance(25);
  Result :=25;
end;

procedure TPartikel.PartikelRender;
var
  i: integer;
begin
  glEnable(GL_BLEND);
  glBlendfunc(GL_SRC_ALPHA, GL_ONE);
  glDepthMask(false);
  for i:=0 to Partikel.partikelnr do
    if Partikel.running[i] then
    begin
      glpushmatrix;
        gltranslated(Partikel.Effects[i].translate.X, Partikel.Effects[i].translate.Y,
Partikel.Effects[i].translate.Z);
        glscaled(Partikel.Effects[i].scale.X, Partikel.Effects[i].scale.Y, Partikel.Effects[i].scale.Z);
        glrotated(Partikel.Effects[i].rotate.angle,Partikel.Effects[i].rotate.X,
Partikel.Effects[i].rotate.Y, Partikel.Effects[i].rotate.Z);
        Partikel.Effects[i].Render;
      glpopmatrix;
    end;
```

```
  glDepthMask(true);
end;

procedure TPartikel.deletepartikel(which: integer);
begin
  Partikel.Effects[which].Destroy;
  running[which]:=false;
end;

initialization
begin
  Partikel :=      TPartikel.Create;
  SDL_Init(SDL_INIT_TIMER);
end;

end.
```

14. pfxCore

```
{
***********************************[ pfxCore ]***********************************

Grundlegende Records und Klassen für den Aufbau von Partikel-Effekten!

(stört euch nicht and den englischen Komentaren... ist bei mir so drin! ;))

*****************************[Version 1.0 22.09.2002]*****************************
}

unit pfxCore;

interface

type

TPfxVector = record
  x     : single;
  y     : single;
  z     : single;
end;

TPfxColor = record
  r     : single;
  g     : single;
  b     : single;
end;

TPfxParticle = record
  Position : TPfxVector;
  Velocity : TPfxVector;
  Density  : single;
  Mass     : single;
```

```
  Size     : single;
  Color    : TPfxColor;
  LiveSpan : integer;
  Age      : integer;
end;

TPfxContainer = class(TObject)
protected
  FnumParticles : word; // used slots in the array of size
  function GetSize : word;
public
  Particles  : array of TPfxParticle; //add Particles with method Add(); don't use index >
numParticles
  property numParticles : word read FnumParticles;
  property Size : word read GetSize;
  constructor Create(aSize : word); overload;
  function Add(var aParticle : TPfxParticle) : integer; //returns Index of created Particle
  procedure Delete(aIndex : integer); // sets Energy := 0
  procedure Advance(aTime : integer); // time in ms
  function Clean : word; //Delete Particles with Energy = 0; Returns number of deleted Particles
  procedure Clear;
end;

procedure FXBillboardBegin;
procedure FXBillboardEnd;

{TPfxSystem = class(TObject)
protected
  function GetAlive : Boolean; virtual; abstract;
public
  property Alive : Boolean read GetAlive;
  procedure Render; virtual; abstract;
  procedure Advance(aTime : integer); virtual; abstract;
end;}

implementation

uses windows, openGL;

procedure FXBillboardBegin;
var
  x,y : byte;
  Matrix : array[0..15] of single;
begin
//delete rotation part - replace it with a 3x3 Identity Matrix
 glGetFloatv(GL_MODELVIEW_MATRIX, @Matrix); //get matrix
 for x := 0 to 2 do
  for y := 0 to 2 do
   if x=y then Matrix[x*4+y] := 1 else Matrix[x*4+y] := 0;
 glLoadMatrixf(@Matrix); //replace modviewmat
end;
```

```
procedure FXBillboardEnd;
begin
  //restore original Matrix
  glPopMatrix;
end;

function TPfxContainer.GetSize;
begin
  result := length(Particles);
end;

constructor TPfxContainer.Create(aSize : word);
begin
  SetLength(Particles, aSize);
  FnumParticles := 0;
end;

function TPfxContainer.Add( var aParticle : TPfxParticle) : integer;
begin
  if FnumParticles = Size-1 then Clean; //Aufräumen, wenn Array voll
  //Partikel hinzufügen.
  Particles[FnumParticles] := aParticle;
  if FnumParticles < Size-1 then inc(FnumParticles);
  result := FnumParticles;
end;

procedure TPfxContainer.Delete(aIndex : integer);
begin
  Particles[aIndex].LiveSpan := 0;
end;

procedure TPfxContainer.Advance(aTime : integer);
var i : integer;
begin
  for i := 0 to FnumParticles-1 do begin
    with Particles[i] do begin
      //S := S_0 + v * t
      Position.X := Position.X + Velocity.X*aTime / 1000;
      Position.Y := Position.Y + Velocity.Y*aTime / 1000;
      Position.Z := Position.Z + Velocity.Z*aTime / 1000;
      LiveSpan := LiveSpan - aTime;
      Age := Age + aTime;
    end;
  end;
end;

function TPfxContainer.Clean : word;
var i, pos : integer;
begin
 pos := 0;
 //copy array into itself - ignore particles with Livespan < 0
 for i := 0 to FNumParticles - 1 do begin
```

```
   Particles[pos] := Particles[i];
   if Particles[pos].LiveSpan > 0 then inc(pos);
 end;
 result := FNumParticles - pos;
 FNumParticles := pos;
end;

procedure TPfxContainer.Clear;
var i : word;
begin
  FnumParticles := 0;
end;

end.
```

15. pfxImp

```
unit pfxImp;

interface

uses pfxCore, dglopengl, textures, sysutils;

type
  TVector = record
    x:      double;
    y:      double;
    z:      double;
    angle:  double;
  end;

  TFXSettings = record
    PATH                  :           string;
    particlenr            :           integer;
    START_LIVESPAN        :           integer;
    EMISSION_RATE         :           integer;
    PARTICLES_PER_EMISSION  :         integer;
    TexPath               :           string;
    agedurch              :           integer;
    ifage                 :           integer;
    randomr               :           boolean;
    randomg               :           boolean;
    randomb               :           boolean;
    colr                  :           double;
    colg                  :           double;
    colb                  :           double;
    randomVelx            :           integer;
    randomVely            :           integer;
    randomVelz            :           integer;
    randomposx            :           integer;
    randomposy            :           integer;
    randomposz            :           integer;
    startrandomVelx       :           integer;
```

```
    startrandomVely         :           integer;
    startrandomVelz         :           integer;
    startrandomposx         :           integer;
    startrandomposy         :           integer;
    startrandomposz         :           integer;
    durchVelx               :           double;
    durchVely               :           double;
    durchVelz               :           double;
    durchposx               :           double;
    durchposy               :           double;
    durchposz               :           double;
    startdurchVelx          :           double;
    startdurchVely          :           double;
    startdurchVelz          :           double;
    startdurchposx          :           double;
    startdurchposy          :           double;
    startdurchposz          :           double;
    plusVelx                :           double;
    plusVely                :           double;
    plusVelz                :           double;
    plusposx                :           double;
    plusposy                :           double;
    plusposz                :           double;
    startplusVelx           :           double;
    startplusVely           :           double;
    startplusVelz           :           double;
    startplusposx           :           double;
    startplusposy           :           double;
    startplusposz           :           double;
  end;

  TExampleFX = class(Tobject)
    Container :             TPfxContainer;
    Particle :              TPfxParticle;
    EmissionTime :          integer;
    FireTex :               GluInt;
    FXSettings:             TFXSettings;
    translate:              TVector;
    rotate:                 TVector;
    scale:                  TVector;
    constructor Create;
    destructor Destroy; override;
    procedure Render;
    procedure Advance(aTime : integer);
    procedure Settings;
  end;

implementation

procedure TExampleFX.Render;
var i : integer;
    sat : single;
```

```
begin
  glbindtexture(GL_TEXTURE_2D, FireTex);
  for i := 0 to Container.numParticles do with Container.Particles[i] do
   if LiveSpan > 0 then begin
    if age > FXSettings.ifage then sat := 1 - Age / FXSettings.START_LIVESPAN else sat := Age /
FXSettings.agedurch;
    //save original Matrix
    glPushMatrix;
    //Set Particle Color and Alpha (sat)
    glColor4f(color.r,color.g,color.b,sat / 2);
    glTranslatef(Position.x,Position.y,Position.z);
    FXBillboardBegin;
    //Render Quad
    glscalef(scale.x, scale.y, scale.z);
    glBegin(GL_QUADS);
      glTexCoord2f(1,0);
      glVertex3f(0.05,-0.05,0);
      glTexCoord2f(0,0);
      glVertex3f(-0.05,-0.05,0);
      glTexCoord2f(0,1);
      glVertex3f(-0.05,0.05,0);
      glTexCoord2f(1,1);
      glVertex3f(+0.05,+0.05,0);
    glEnd;
    glPopMatrix;
  end;
end;

procedure TExampleFX.Advance(aTime : integer);
var i : integer;
begin
  inc(EmissionTime, aTime);
  while emissionTime > FXSettings.EMISSION_RATE do with Particle do begin
    //Set Particle-Template
    Position.x := random(FXSettings.startrandomPosx)/FXSettings.startdurchPosx +
FXSettings.startplusPosx;
    Position.y := random(FXSettings.startrandomPosy)/FXSettings.startdurchPosy +
FXSettings.startplusPosy;
    Position.z := random(FXSettings.startrandomPosz)/FXSettings.startdurchPosz +
FXSettings.startplusPosz;
    Velocity.x := random(FXSettings.startrandomVelx)/FXSettings.startdurchVelx +
FXSettings.startplusVelx;
    Velocity.y := random(FXSettings.startrandomVely)/FXSettings.startdurchVely +
FXSettings.startplusVely;
    Velocity.z := random(FXSettings.startrandomVelz)/FXSettings.startdurchVelz +
FXSettings.startplusVelz;

    if FXSettings.randomr then Color.r := random * FXSettings.colr
    else Color.r:=FXSettings.colr;
    if FXSettings.randomg then Color.g := random * FXSettings.colg
    else Color.g:=FXSettings.colg;
    if FXSettings.randomb then Color.b := random * FXSettings.colb
```

```
    else Color.b:=FXSettings.colb;

    LiveSpan := FXSettings.START_LIVESPAN;
    //emmit Particles based on the Template
    for i := 1 to FXSettings.PARTICLES_PER_EMISSION do begin
      Position.x := Position.x + random(FXSettings.randomPosx)/FXSettings.durchPosx +
FXSettings.plusPosx;
      Position.y := Position.y + random(FXSettings.randomPosy)/FXSettings.durchPosy +
FXSettings.plusPosy;
      //Position.z := Position.z + random(FXSettings.randomPosz)/FXSettings.durchPosz +
FXSettings.plusPosz;
      Velocity.x := Velocity.x + random(FXSettings.randomVelx)/FXSettings.durchVelx +
FXSettings.plusVelx;
      Velocity.y := Velocity.y + random(FXSettings.randomVely)/FXSettings.durchVely +
FXSettings.plusVely;
      //Velocity.z := Velocity.z + random(FXSettings.randomVelz)/FXSettings.durchVelz +
FXSettings.plusVelz;
      Container.Add(Particle);
    end;
    dec(EmissionTime, FXSettings.EMISSION_RATE);
  end;
  //Update all existing Particles
  Container.Advance(aTime);
end;

constructor TExampleFX.Create;
begin
  inherited;
end;

destructor TExampleFX.Destroy;
begin
  container.free;
  inherited;
end;

procedure TExampleFX.Settings;
begin
  with FXSettings do
  begin
    particlenr            :=    600;
    START_LIVESPAN        :=    1000;
    EMISSION_RATE         :=    2;
    PARTICLES_PER_EMISSION  :=    1;
    TexPath               :=    'Fire.tga';
    agedurch              :=    100;
    ifage                 :=    100;
    randomr               :=    false;
    randomg               :=    true;
    randomb               :=    true;
    colr                  :=    1;
    colg                  :=    0.7;
```

```
    colb                := 0.25;
    randomPosx          := 10;
    randomPosy          := 5;
    randomPosz          := 1;
    randomVelx          := 4;
    randomVely          := 6;
    randomVelz          := 1;
    startrandomposx     := 20;
    startrandomposy     := 5;
    startrandomposz     := 5;
    startrandomVelx     := 60;
    startrandomVely     := 80;
    startrandomVelz     := 40;
    durchPosx           := 100;
    durchPosy           := 100;
    durchPosz           := 100;
    durchVelx           := 100;
    durchVely           := 100;
    durchVelz           := 100;
    startdurchPosx      := 100;
    startdurchPosy      := 100;
    startdurchPosz      := 100;
    startdurchVelx      := 100;
    startdurchVely      := 100;
    startdurchVelz      := 100;
    plusPosx            := -0.05;
    plusPosy            := 0;
    plusPosz            := 0;
    plusVelx            := -0.02;
    plusVely            := 0.03;
    plusVelz            := 0;
    startplusPosx       := -0.1;
    startplusPosy       := 0;
    startplusPosz       := 0;
    startplusVelx       := -0.3;
    startplusVely       := 0.2;
    startplusVelz       := -0.2;
  end;
  FXSettings.startplusPosz      := 0;
end;

end.
```

16. UCharacters

```
unit UCharacters;

interface

uses
  gl3ds,
  Basics,
  dglOpenGL,
```

```
  PartikelUnit,
  Textures,
  GLFrustum,
  IniFiles,
  ULogger,
  SysUtils,
  GUI;

type
  TCharacters = class;

  TCharPistol = record
    Pistol:          integer;
    magazines:       integer;
    shots:           integer;
    lastShot:        Cardinal;
  end;

  TCharRifle = record
    Rifle:           integer;
    magazines:       integer;
    shots:           integer;
    lastShot:        Cardinal;
  end;

  TEquip = class(TObject)
    description:     string;
    name:            string;
    prize:           integer;
    graphic:         cardinal;
    constructor Create(_Name: string);
    destructor Destroy; override;
  end;

  TWeapon = class(TObject)
    Weapondmg:       single;       //sollte etwa 20 sein, bei einem sturmgewehr...
    accuracy:        single;       //genauigkeit
    prize:           integer;      //preis für die waffe

    magazineshots:   integer;      //ca 30
    maxmagazines:    integer;      //ca 3
    magazineprize:   integer;      //ca 100
    sleep:           integer;      //ca 30ms -> 300
    reloadsleep:     integer;      //ca 100ms  -> 1000

    graphic:         cardinal;     //2d grafik laden müsste man irgendwann noch
    description:     string;       //wichtig für die GUI...
    name:            string;       //Name der Waffe (Ordnername)
    mesh:            TAll3dsMesh;  //evtl später einfügen für 3d grafik
    function CalcDMG(Factor: single): integer;
    constructor Create(_Name: string);
    destructor Destroy; override;
```

```
  procedure Render;
 end;

 TWeapons = class(TObject)
  Pistols:        array of TWeapon;
  Rifles:         array of TWeapon;
  Equips:          array of TEquip;
  function AddPistol(name: string): integer;
  function AddRifle(name: string): integer;
  constructor Create;
  destructor Destroy; override;
 end;

 TCharacter = class(TObject)
 private
  _Dead:            boolean;
  _Armor:           integer;
  _Name:            string;
  _Terrorist:       boolean;
  _ModelNr:          integer;
  _Health:          integer;
  _used:            boolean;
  _Kills:           integer;
  _Deaths:          integer;

  //Weapons:
  _currentweaponslot:integer;

  procedure SetArmor(Value: integer);
  procedure SetDead(Value: boolean);
  procedure SetDeaths(Value: integer);
  procedure SetHealth(Value: integer);
  procedure SetKills(Value: integer);
  procedure SetModelNr(Value: integer);
  procedure SetName(Value: string);
  procedure SetTerrorist(Value: boolean);
  procedure SetUsed(Value: boolean);
  procedure SetWeaponslot(Value: integer);
 public
  //Position
  Position:         TPlayerPosition;

  //Mesh
  Mesh:             TAll3dsMesh;

  //Bot or Not, important for the KIunit.
  Bot:              boolean;

  //Game things
  Dollar:           integer;
  HeadHeight:       single;     //where the Camera is.
```

```
  //for Collisions
  Sphere:          TSpheref;
  HeadSphere:      TSpheref;
  ChestSphere:     TSpheref;
  LegSphere:       TSpheref;

  //Weapons
  Pistol:          TCharPistol;
  Rifle:           TCharRifle;

  Index:           Integer;

  Owner:           TCharacters;

  procedure buyPistol(Index: integer);
  procedure buyRifle(Index: integer);
  procedure buyEquip(Index: integer);
  procedure CalcDropDmg(Speed: single);
  constructor Create(_Index, Model: integer; _Owner: TCharacters);
  destructor Destroy; override;
  function GetRay: TRayf;
  procedure Move(x,y,z: single);  overload;
  procedure Move(v: TVector3f);   overload;
  procedure MoveSphere(v: TVector3f); overload;
  procedure MoveSphere(x,y,z: single); overload;
  procedure PlusMove(PlusX, PlusY, PlusZ: single); overload;
  procedure PlusMove(PlusV: TVector3f); overload;
  procedure Reload;
  procedure Render;
  procedure HitLeftMouse;
  procedure Shoot(const Ray: TRayf; Weapon: TWeapon);

  property Armor: integer            read _Armor            write setArmor;
  property Currentweaponslot: integer read _CurrentWeaponslot write setWeaponslot;
  property Dead: boolean             read _Dead             write setDead;
  property Deaths: integer           read _Deaths           write setDeaths;
  property Health: integer           read _Health           write setHealth;
  property Kills: integer            read _Kills            write setKills;
  property ModelNr: integer          read _ModelNr          write setModelNr;
  property Name: string              read _Name             write setName;
  property Terrorist: boolean        read _Terrorist        write setTerrorist;
  property Used: boolean             read _Used             write setUsed;
end;

TCharacters = class(TObject)
private
  _LivingPlayers:   integer;
  _ownChar:         integer;
  procedure _SetLivingPlayers(Value: Integer);
  procedure _setOwnChar(Value: integer);
public
  char:            array of TCharacter;
```

```
    CurrentChar:      TCharacter;

    numPlayers:       integer;
    numChars:         integer;
    numTerrors:       integer;
    numCTs:           integer;
    numBots:          integer;
    Quadric:          PGLUQuadric;  //Zum Anzeigen der Kugeln.
    NextRespawn:      integer;      //seconds/1000
    NewRespawn:       boolean;

    WaitforPlayers:   boolean;

    function  AddChar(x,y,z: double; Model: integer): integer;
    function  CalcLivingPlayers: integer;
    constructor Create;
    destructor Destroy; override;
    procedure RearrangeCharArray;
    procedure RemoveChar(_Char: integer);
    procedure RenderChars;
    procedure Respawn;

    property LivingPlayers: integer read _LivingPlayers write _setLivingPlayers;
    property ownChar: integer       read _ownChar       write _setOwnChar;
  end;

  function CheckShot(const Ray: TRayf; const Enemy: TCharacter): integer;

var
  Chars:  TCharacters;
  Weapons: TWeapons;

implementation

uses
  Mainunit,
  GUIAdds,
  UNetwork,
  SDL_Net,
  EventHandlerUnit,
  ULevels;

//returns integer:
//0: not hit
//1: head
//2: chest & arms
//3: legs
function CheckShot(const Ray: TRayf; const Enemy: TCharacter): integer;
var
  i,j: integer;
  dist: single;
begin
```

```
  result := 0;
  if not Col_Sphere_Ray(Enemy.Sphere, Ray) then
    Exit;

  //check head
  if Col_Sphere_Ray(Enemy.HeadSphere, Ray) then
  begin
    result := 1;
  end;

  //check Chest
  if Col_Sphere_Ray(Enemy.ChestSphere, Ray) then
  begin
    result := 2;
  end;

  //check legs
  if Col_Sphere_Ray(Enemy.LegSphere, Ray) then
  begin
    result := 3;
  end;

  if result <> 0 then
  begin
    dist := Magnitude(MinusVector(Enemy.Sphere.center, Ray.Origin));
    with game.CurrentLevel.Octree do
      for i := 0 to High(Nodes) do
        //optimieren, alle Dreiecke werden abgefragt, sollten es aber nicht werden.
        for j := 0 to Nodes[i].numv - 1 do
          if Col_Triangle_Ray(game.CurrentLevel.Octree.Nodes[i].poly[j], Ray, dist) then
          begin
            result := 0;
            Exit;
          end;
  end;
end;

{-------------------------------------------------------------------------------
 --------------------           TEquip            -------------------------
 -------------------------------------------------------------------------------}
constructor TEquip.Create(_Name: string);
var
  ini: TIniFile;
  fname: string;
begin
  inherited Create;
  fname := 'data\weapons\' + _name + '\';
  try
    ini          := TIniFile.Create(fname + 'config.ini');
    prize        := ini.ReadInteger( 'config', 'prize',         2000);
    description  := ini.ReadString(  'config', 'description',   'default description. change plz.');
  finally
```

```
    ini.Free;
  end;
  name := _name;

  //load 2d graphic...
  LoadTexture(fname + 'graphic.jpg', graphic, false);
end;

destructor TEquip.Destroy;
begin
  gldeletetextures(1,@graphic);
  inherited Destroy;
end;
{-------------------------------------------------------------------------------
 --------------------          TWeapon          -------------------------
 -------------------------------------------------------------------------------}
function TWeapon.CalcDMG(Factor: single): integer;
begin
  result := round(Factor*WeaponDMG - 5 + random(10));
end;

constructor TWeapon.Create(_Name: string);
var
  ini: TIniFile;
  fname: string;
begin
  inherited Create;
  Mesh := nil;

  fname := 'data\weapons\' + _name + '\';
  //load graphic...
  if Fileexists(fname + 'weapon.3ds') then
  begin
    mesh := TAll3dsMesh.Create(nil);
    mesh.TexturePath := fname;
    mesh.LoadFromFile(fname + 'weapon.3ds');
    //mesh.BuildDisplayList;
  end
  else
    Log.AddError('Failed to load Weapon: ' + fname, 'UCharacters');

  //load 2d graphic...
  LoadTexture(fname + 'graphic.jpg', graphic, false);

  //load settings
  try
    ini          := TIniFile.Create(fname + 'config.ini');
    Weapondmg    := ini.ReadFloat(   'config', 'Weapondmg',      20);
    Accuracy     := ini.ReadFloat(   'config', 'accuracy',       1.0);
    magazineshots := ini.ReadInteger( 'config', 'magazineshots',  30);
    maxmagazines  := ini.ReadInteger( 'config', 'maxmagazines',   3);
    prize        := ini.ReadInteger( 'config', 'prize',          2000);
```

```
    description   := ini.ReadString(  'config', 'description',    'default description. change plz.');
    magazineprize := ini.ReadInteger( 'config', 'magazineprize',  100);
    sleep         := ini.ReadInteger( 'config', 'sleep',          300);       // = 0.3s
    reloadsleep   := ini.ReadInteger( 'config', 'reloadsleep',    1000);      // = 1s
  finally
    ini.Free;
  end;
  name := _name;
end;

destructor TWeapon.Destroy;
var
  name: string;
begin
  mesh.Destroy;
  gldeletetextures(1,@graphic);
  inherited Destroy;
end;

procedure TWeapon.Render;
begin
  if Mesh <> nil then
  begin
    //glTranslatef(-0.1419, -0.1885, 0.5525);
    //Daten von 3ds Max
    glTranslatef(-0.1032, -0.1698, 0.6012);
    glRotatef(5, -0.5,1,0);
    Mesh.Render;
  end;
end;

{-------------------------------------------------------------------------------
 --------------------          TWeapons          -------------------------
 -------------------------------------------------------------------------------}
function TWeapons.AddPistol(name: string): integer;
begin
  setlength(Pistols, Length(Pistols) + 1);
  Pistols[High(Pistols)] := TWeapon.Create(name);
  result := High(Pistols);
end;

function TWeapons.AddRifle(name: string): integer;
begin
  setlength(Rifles, Length(Rifles) + 1);
  Rifles[High(Rifles)] := TWeapon.Create(name);
  result := High(Rifles);
end;

constructor TWeapons.Create;
var
  ini: TIniFile;
  i: integer;
```

```
begin
  inherited Create;
  //load settings
  try
    ini := TIniFile.Create('data\weapons\config.ini');
    for i := 1 to 5 do
      AddPistol(ini.ReadString('pistols', 'Pistol'+inttostr(i), 'default'));
    for i := 1 to 5 do
      AddRifle(ini.ReadString('rifles', 'Rifle'+inttostr(i), 'default'));
    for i := 1 to 5 do
    begin
      setlength(Equips, Length(Equips) + 1);
      Equips[High(Equips)] := TEquip.Create(ini.ReadString('equips', 'Equip'+inttostr(i), 'default'));
    end;
  finally
    ini.Free;
  end;
end;

destructor TWeapons.Destroy;
var
  i: integer;
begin
  for i := 0 to High(Pistols) do
    Pistols[i].Destroy;
  for i := 0 to High(Rifles) do
    Rifles[i].Destroy;
  inherited Destroy;
end;

{-------------------------------------------------------------------------------
 --------------------          TCharacter          -------------------------
 -------------------------------------------------------------------------------}
procedure TCharacter.buyPistol(Index: integer);
begin
  if dollar >= Weapons.Pistols[Index].prize then
  begin
    dollar := dollar - Weapons.Pistols[Index].prize;
    Console.AddString('You successfully bought a ' + Weapons.Pistols[Index].name);
    GUIclass.Windows[wBuy].Visible := false;
    Pistol.Pistol := Index;
    Pistol.magazines := 0;
    Pistol.shots     := Weapons.Pistols[Index].magazineshots;
    Net.hasInfos     := true;
  end
  else
    Console.AddString('You haven''t enough money to buy this pistol!');
end;

procedure TCharacter.buyRifle(Index: integer);
begin
  if dollar >= Weapons.Rifles[Index].prize then
```

```
  begin
    dollar := dollar - Weapons.Rifles[Index].prize;
    Console.AddString('You successfully bought a ' + Weapons.Rifles[Index].name);
    GUIclass.Windows[wBuy].Visible := false;
    Rifle.Rifle := Index;
    Rifle.magazines := 0;
    Rifle.magazines := 0;
    Rifle.shots     := Weapons.Rifles[Index].magazineshots;
    Net.hasInfos    := true;
  end
  else
    Console.AddString('You haven''t enough money to buy this rifle!');
end;

procedure TCharacter.buyEquip(Index: integer);
begin
  case Index of
    0: begin
        //magazines
        case currentweaponslot of
          0: if Rifle.Rifle > -1 then
             begin
               if dollar >= Weapons.Rifles[Rifle.Rifle].magazineprize then
               begin
                 if Weapons.Rifles[Rifle.Rifle].maxmagazines <= Rifle.magazines then
                   Console.AddString('Your magazines are full.')
                 else
                 begin
                   dollar := dollar - Weapons.Rifles[Rifle.Rifle].magazineprize;
                   inc(Rifle.magazines);
                   Console.AddString('You bought a magazine.');
                 end;
               end
               else
                Console.AddString('You have not enough money.');
             end
             else
               Console.AddString('Buy a Weapon first!');

          1: if Pistol.Pistol > -1 then
             begin
               if dollar >= Weapons.Pistols[Pistol.Pistol].magazineprize then
               begin
                 if Weapons.Pistols[Pistol.Pistol].maxmagazines <= Pistol.magazines then
                   Console.AddString('Your magazines are full.')
                 else
                 begin
                   dollar := dollar - Weapons.Pistols[Pistol.Pistol].magazineprize;
                   inc(Pistol.magazines);
                   Console.AddString('You bought a magazine.');
                 end;
               end
```

```
            else
              Console.AddString('You have not enough money.');
            end
            else
              Console.AddString('Buy a Weapon first!');

        else
          Console.AddString('You have not armed any weapon.');

        end;
      end;
    //armor
    1: begin
        if dollar >= Weapons.Equips[1].prize then
        begin
          dollar := dollar - Weapons.Equips[1].prize;
          Owner.CurrentChar.Armor := 100;
          Console.AddString('You bought armour.');
        end
        else
          Console.AddString('You have not enough money.');
      end;
    2: begin
        //explosives?!

      end;
    3: begin
        //smokegrandades?!

      end;
    4: begin
        //nightsight?!

      end;
  end;
end;

procedure TCharacter.CalcDropDmg(Speed: single);
begin
  if (speed < -6.5) then //bei mehr als 6.5 m/s nimmt die figur schaden. ca. 2m höhe.
    Health := Health - round((-speed - 6.5) ); //* 8);              //bei ca. 8m+ fall stirbt er.
end;

constructor TCharacter.Create(_Index, Model: integer; _Owner: TCharacters);
begin
  inherited Create;
  Mesh  := nil;
  Index := _Index;
  used  := true;
  _ModelNr := -1;
  Owner := _Owner;
```

```
  ModelNr := Model;

  //set defaults:
  position.x := 0;
  position.y := 0;
  position.z := 0;
  health     := 0;
  armor      := 0;
  bot        := false;
  dollar     := 16000;
  dead       := true;

  //Weapons
  Rifle.Rifle       := -1;
  Rifle.magazines   := 0;
  Rifle.shots       := 0;
  Rifle.lastShot    := 0;

  Pistol.Pistol     := 0;
  Pistol.magazines  := 0;
  Pistol.shots      := 0;
  Pistol.lastShot   := 0;
end;

destructor TCharacter.Destroy;
begin
  mesh.Free;
  inherited Destroy;
end;

function TCharacter.GetRay: TRayf;
begin
  //ADD die ungenauigkeit der Waffe!

  Result.Origin.x := Position.x;
  Result.Origin.y := HeadSphere.center.y;
  Result.Origin.z := Position.z;

  Result.Direction.x := cos(Position.lookanglex*DEGTORAD);
  Result.Direction.y := -sin(Position.lookangley*DEGTORAD);   //- braucht es kA warum.
  Result.Direction.z := sin(Position.lookanglex*DEGTORAD);

  Result.Direction := Normalize(Result.Direction);
end;

procedure TCharacter.HitLeftMouse;
var
  Package: TTCPPackage;
begin
  case currentweaponslot of
    //Rifle
    0: begin
```

```
        if (Rifle.Rifle > -1) then
          if ((Rifle.lastShot + Weapons.Rifles[Rifle.Rifle].sleep - EventHandler.counter) < 0)
             and (EventHandler.lastReload < (EventHandler.counter -
Weapons.Rifles[Rifle.Rifle].reloadsleep)) then
            if Rifle.shots > 0 then
            begin
              Rifle.lastShot := EventHandler.counter;
              if Net.IsServer then
                Shoot(GetRay, Weapons.Rifles[Rifle.Rifle])
              else
                if Net.IsClient then
                begin
                  package.MessageType := nClient_Shoot;
                  package.Shoot := GetRay;
                  PackCollector.AddPacket(Net.Client.TCPSocket, Package, SizeOf(TTCPPackage));
                  //SDLNet_TCP_Send(Net.Client.TCPSocket, @package, SizeOf(TTCPPackage));
                end;
              dec(Rifle.shots);
            end;
      end;
    //Pistol
    1: begin
        if (Pistol.Pistol > -1) then
          if (Pistol.lastShot + Weapons.Pistols[Pistol.Pistol].sleep - EventHandler.counter < 0)
             and (EventHandler.lastReload < EventHandler.counter -
Weapons.Pistols[Pistol.Pistol].reloadsleep) then
            if Pistol.shots > 0 then
            begin
              Pistol.lastShot := EventHandler.counter;
              if Net.IsServer then
                Shoot(GetRay, Weapons.Rifles[Pistol.Pistol])
              else
               if Net.IsClient then
               begin
                  package.MessageType := nClient_Shoot;
                  package.Shoot := GetRay;
                  PackCollector.AddPacket(Net.Client.TCPSocket, Package, SizeOf(TTCPPackage));
                  //SDLNet_TCP_Send(Net.Client.TCPSocket, @package, SizeOf(TTCPPackage));
               end;
              dec(Pistol.shots);
            end;
      end;
    //Knife
    2: begin

      end;
    //Grenades
    3: begin

      end;

  end;
```

```
end;

procedure TCharacter.Move(x,y,z: single);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + x - Position.x;
  HeadSphere.center.y := HeadSphere.center.y + y - Position.y;
  HeadSphere.center.z := HeadSphere.center.z + z - Position.z;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + x - Position.x;
  ChestSphere.center.y := ChestSphere.center.y + y - Position.y;
  ChestSphere.center.z := ChestSphere.center.z + z - Position.z;

  //Legs
  LegSphere.center.x := LegSphere.center.x + x - Position.x;
  LegSphere.center.y := LegSphere.center.y + y - Position.y;
  LegSphere.center.z := LegSphere.center.z + z - Position.z;

  //Sphere
  Sphere.center.x := Sphere.center.x + x - Position.x;
  Sphere.center.y := Sphere.center.y + y - Position.y;
  Sphere.center.z := Sphere.center.z + z - Position.z;

  //Position
  Position.x := x;
  Position.y := y;
  Position.z := z;
end;

procedure TCharacter.Move(v: TVector3f);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + v.x - Position.x;
  HeadSphere.center.y := HeadSphere.center.y + v.y - Position.y;
  HeadSphere.center.z := HeadSphere.center.z + v.z - Position.z;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + v.x - Position.x;
  ChestSphere.center.y := ChestSphere.center.y + v.y - Position.y;
  ChestSphere.center.z := ChestSphere.center.z + v.z - Position.z;

  //Legs
  LegSphere.center.x := LegSphere.center.x + v.x - Position.x;
  LegSphere.center.y := LegSphere.center.y + v.y - Position.y;
  LegSphere.center.z := LegSphere.center.z + v.z - Position.z;

  //Sphere
  Sphere.center.x := Sphere.center.x + v.x - Position.x;
  Sphere.center.y := Sphere.center.y + v.y - Position.y;
  Sphere.center.z := Sphere.center.z + v.z - Position.z;
```

```
  //Position
  Position.x := v.x;
  Position.y := v.y;
  Position.z := v.z;
end;

procedure TCharacter.MoveSphere(v: TVector3f);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + v.x - Sphere.center.x;
  HeadSphere.center.y := HeadSphere.center.y + v.y - Sphere.center.y;
  HeadSphere.center.z := HeadSphere.center.z + v.z - Sphere.center.z;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + v.x - Sphere.center.x;
  ChestSphere.center.y := ChestSphere.center.y + v.y - Sphere.center.y;
  ChestSphere.center.z := ChestSphere.center.z + v.z - Sphere.center.z;

  //Legs
  LegSphere.center.x := LegSphere.center.x + v.x - Sphere.center.x;
  LegSphere.center.y := LegSphere.center.y + v.y - Sphere.center.y;
  LegSphere.center.z := LegSphere.center.z + v.z - Sphere.center.z;

  //Position
  Position.x := Position.x + v.x - Sphere.center.x;
  Position.y := Position.y + v.y - Sphere.center.y;
  Position.z := Position.z + v.z - Sphere.center.z;

  //Sphere
  Sphere.center.x := v.x;
  Sphere.center.y := v.y;
  Sphere.center.z := v.z;
end;

procedure TCharacter.MoveSphere(x,y,z: single);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + x - Sphere.center.x;
  HeadSphere.center.y := HeadSphere.center.y + y - Sphere.center.y;
  HeadSphere.center.z := HeadSphere.center.z + z - Sphere.center.z;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + x - Sphere.center.x;
  ChestSphere.center.y := ChestSphere.center.y + y - Sphere.center.y;
  ChestSphere.center.z := ChestSphere.center.z + z - Sphere.center.z;

  //Legs
  LegSphere.center.x := LegSphere.center.x + x - Sphere.center.x;
  LegSphere.center.y := LegSphere.center.y + y - Sphere.center.y;
  LegSphere.center.z := LegSphere.center.z + z - Sphere.center.z;

  //Position
```

```
  Position.x := Position.x + x - Sphere.center.x;
  Position.y := Position.y + y - Sphere.center.y;
  Position.z := Position.z + z - Sphere.center.z;

  //Sphere
  Sphere.center.x := x;
  Sphere.center.y := y;
  Sphere.center.z := z;
end;

procedure TCharacter.PlusMove(PlusX, PlusY, PlusZ: single);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + PlusX;
  HeadSphere.center.y := HeadSphere.center.y + PlusY;
  HeadSphere.center.z := HeadSphere.center.z + PlusZ;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + PlusX;
  ChestSphere.center.y := ChestSphere.center.y + PlusY;
  ChestSphere.center.z := ChestSphere.center.z + PlusZ;

  //Legs
  LegSphere.center.x := LegSphere.center.x + PlusX;
  LegSphere.center.y := LegSphere.center.y + PlusY;
  LegSphere.center.z := LegSphere.center.z + PlusZ;

  //Sphere
  Sphere.center.x := Sphere.center.x + PlusX;
  Sphere.center.y := Sphere.center.y + PlusY;
  Sphere.center.z := Sphere.center.z + PlusZ;

  //Position
  Position.x := Position.x + PlusX;
  Position.y := Position.y + PlusY;
  Position.z := Position.z + PlusZ;
end;

procedure TCharacter.PlusMove(PlusV: TVector3f);
begin
  //Head
  HeadSphere.center.x := HeadSphere.center.x + PlusV.X;
  HeadSphere.center.y := HeadSphere.center.y + PlusV.Y;
  HeadSphere.center.z := HeadSphere.center.z + PlusV.Z;

  //Chest
  ChestSphere.center.x := ChestSphere.center.x + PlusV.X;
  ChestSphere.center.y := ChestSphere.center.y + PlusV.Y;
  ChestSphere.center.z := ChestSphere.center.z + PlusV.Z;

  //Legs
  LegSphere.center.x := LegSphere.center.x + PlusV.X;
```

```
  LegSphere.center.y := LegSphere.center.y + PlusV.Y;
  LegSphere.center.z := LegSphere.center.z + PlusV.Z;

  //Sphere
  Sphere.center.x := Sphere.center.x + PlusV.X;
  Sphere.center.y := Sphere.center.y + PlusV.Y;
  Sphere.center.z := Sphere.center.z + PlusV.Z;

  //Position
  Position.x := Position.x + PlusV.X;
  Position.y := Position.y + PlusV.Y;
  Position.z := Position.z + PlusV.Z;
end;

procedure TCharacter.Reload;
begin
  case Currentweaponslot of
    0:  begin
         if (Rifle.magazines > 0) and (Rifle.Rifle >= 0) then
         begin
           dec(Rifle.magazines);
           Rifle.shots := Weapons.Rifles[Rifle.Rifle].magazineshots;
           EventHandler.lastReload := EventHandler.counter;
         end;
        end;

    1:  begin
         if (Pistol.magazines > 0) and (Pistol.Pistol >= 0) then
         begin
           dec(Pistol.magazines);
           Pistol.shots := Weapons.Pistols[Pistol.Pistol].magazineshots;
           EventHandler.lastReload := EventHandler.counter;
         end;
        end;
  end;
end;

procedure TCharacter.Render;
begin
  if ModelNr >= 0 then
  //if not dead and (ModelNr >= 0) then
    //if Frustum.IsSphereWithin(Sphere.center.X,Sphere.Center.Y,Sphere.Center.Z,Sphere.radius)
or (Owner.CurrentChar=self) then
    begin
      glPushMatrix;
        glTranslatef(position.x, position.y{headsphere.center.y}, position.z);
        glRotatef(-position.lookanglex+90,0,1,0);

        //Bei der eigenen Figur anders drehen, da die Waffe und die Hand richtig stehen soll.
        if (Index = Owner.ownChar) and (game.gameVar.cammode = 0) then
        begin
          glRotatef(position.lookangley, 1,0,0);
```

```
      end;
      mesh.Render;

      //Render Weapon
      glPushMatrix;
      //gltranslatef(2,0,0);
      case CurrentWeaponSlot of
        0: if Rifle.Rifle >= 0 then Weapons.Rifles[Rifle.Rifle].Render;
        1: if Pistol.Pistol >= 0 then Weapons.Pistols[Pistol.Pistol].Render;
      //2: Weapons.Knife.Render;
      end;
      glPopMatrix;

      if ULevels.SHOWNODES then
      begin
        //render the Spheres:
        glPushMatrix;
          glTranslatef(0,Sphere.center.y-position.y,0);
          gluSphere(Owner.Quadric,Sphere.radius,20,20);
        glPopMatrix;
        glPushMatrix;
          glTranslatef(0,LegSphere.center.y-position.y,0);
          gluSphere(Owner.Quadric,LegSphere.radius,20,20);
        glPopMatrix;
        glPushMatrix;
          glTranslatef(0,ChestSphere.center.y-position.y,0);
          gluSphere(Owner.Quadric,ChestSphere.radius,20,20);
        glPopMatrix;
        glPushMatrix;
          glTranslatef(0,HeadSphere.center.y-position.y,0);
          gluSphere(Owner.Quadric,HeadSphere.radius,20,20);
        glPopMatrix;
      end;
    glPopMatrix;
  end;
end;

procedure TCharacter.Shoot(const Ray: TRayf; Weapon: TWeapon);
var
  i: integer;
  DMG: integer;
  Hit: integer;
begin
  for i := 0 to High(Owner.char) do
    if (Owner.char[i] <> self) and not Owner.char[i].dead then
      if ((Owner.char[i].Terrorist <> Terrorist ) and not Net.Server.FriendlyFire) or
Net.Server.FriendlyFire then
      begin
        Hit := CheckShot(Ray, Owner.char[i]);          // 3             2      1
        case Hit of                                  //dmg legs: 2.5, chest: 3, head: 7
          //Head
```

```
        1: begin
             DMG := Weapon.CalcDMG(6);
             if random(Armor) > 65 then  //50
               DMG := 0;
             Console.AddString('Hit, Head');
           end;
        //Chest
        2: begin
             DMG := Weapon.CalcDMG(2.5);
             DMG := round(DMG - (DMG*Owner.char[i].Armor/200));
             Owner.char[i].Armor := random(DMG);
             //NET
             if Net.IsServer then
               if Owner.char[i].Index > 0 then
                 Net.Server.Clients[Owner.char[i].Index - 1].SendTCPPacket(nServer_ArmorChange,
inttostr(Owner.char[i].Armor), '', '');
             //NET END
             Console.AddString('Hit, Chest');
           end;
        //Legs
        3: begin
             DMG := Weapon.CalcDMG(1);
             Console.AddString('Hit, Legs');
           end;
       end;
       if Hit > 0 then
       begin
         Owner.char[i].Health := Owner.char[i].Health - DMG;
         if Owner.char[i].Dead then
           Kills := Kills + 1;
         Console.AddString(inttostr(DMG)+' DMG!');
         //Net
         if Net.IsServer then
           if Owner.char[i].Index > 0 then
             Net.Server.Clients[Owner.char[i].Index - 1].SendTCPPacket(nServer_HealthChange,
inttostr(Owner.char[i].Health), '', '');
       end;
     end;
end;

procedure TCharacter.SetArmor(Value: integer);
begin
  if _Armor <> Value then
  begin
    _Armor := Value;
    if Net.IsClient and (Index = Owner.ownChar) then
      Net.Client.hasInfos := true
    else
      if Net.IsServer then
      begin
        if Index > 0 then
        begin
```

```
        Net.Server.Clients[Index-1].hasInfos := true
      end
      else
        Net.Server.hasInfos := true;
    end;
  end;
end;

procedure TCharacter.SetDead(Value: boolean);
begin
  if Value <> _Dead then
  begin
    //if not _Dead and Value then
      //partikel.addpartikel(1,position.x,position.y,position.z,    5,5,5,    1,1,1,0);

    _Dead := Value;
    if Value then
    begin
      Movement.CurrentYSpeed := 0;
      _health := 0;
      armor  := 0;

      Rifle.Rifle      := -1;
      Rifle.magazines   := 0;
      Rifle.shots       := 0;
      Rifle.lastShot    := 0;

      Pistol.Pistol     := 2;
      Pistol.magazines  := 0;
      Pistol.shots      := 0;
      Pistol.lastShot   := 0;
      Deaths := Deaths + 1;
    end
    else
    begin
      Movement.CurrentYSpeed := 0;
    end;
    Owner.CalcLivingPlayers;
  end;
end;

procedure TCharacter.SetDeaths(Value: integer);
begin
  _Deaths := Value;
  if Net.IsServer then
    Net.Server.SendTCPPacket(nServer_StatChange, inttostr(Index), inttostr(Kills),
inttostr(Deaths));
end;

procedure TCharacter.SetHealth(Value: integer);
begin
  if _Health <> Value then
```

```
  begin
    _Health := Value;
    if Health <= 0 then
    begin
      _Health := 0;
      Dead := true;
    end
    else
      dead := false
  end;
end;

procedure TCharacter.SetKills(Value: integer);
begin
  _Kills := Value;
  if Net.IsServer then
    Net.Server.SendTCPPacket(nServer_StatChange, inttostr(Index), inttostr(_Kills),
inttostr(Deaths));
end;

procedure TCharacter.SetModelNr(Value: integer);
var
  xmax,xmin,ymax,ymin,zmax,zmin: single;
  i,j,k: integer;
begin
  if _ModelNr <> Value then
  begin
    dead := true;
    _ModelNr := Value;
    if Net.IsClient then
    begin
      if (Index = Owner.ownChar) then
        Net.Client.hasInfos := true
    end
    else
      if Net.IsServer then
      begin
        if Index > 0 then
          Net.Server.Clients[Index-1].hasInfos := true
        else
          Net.Server.hasInfos := true;
      end;

    //mesh.Free; -> wieder einführen.

    position.x := 0;
    position.y := 0;
    position.z := 0;

    if Value = - 1 then
    begin
      terrorist := false;
```

```
    Sphere.radius         := 0;
    Sphere.Center.x       := 0;
    Sphere.Center.y       := 0;
    Sphere.Center.z       := 0;

    HeadSphere.Center.x   := 0;
    HeadSphere.Center.y   := 0;
    HeadSphere.Center.z   := 0;
    HeadSphere.radius     := 0;

    ChestSphere.Center.x  := 0;
    ChestSphere.Center.y  := 0;
    ChestSphere.Center.z  := 0;
    ChestSphere.radius    := 0;

    LegSphere.Center.x    := 0;
    LegSphere.Center.y    := 0;
    LegSphere.Center.z    := 0;
    LegSphere.radius      := 0;
  end
  else
  begin
    //load mesh
    //0-9 reserved for Police
    //10-19 reserved for Terrorist
    case ModelNr of
      //civil man
      0: begin
           mesh:=TAll3DSMesh.Create(nil);
           mesh.TexturePath:='data\characters\' + sChars[0] + '\';
           mesh.LoadFromFile('data\characters\' + sChars[0] + '\char.3ds');
         end;

      1: begin
           mesh:=TAll3DSMesh.Create(nil);
           mesh.TexturePath:='data\characters\' + sChars[1] + '\';
           mesh.LoadFromFile('data\characters\' + sChars[1] + '\char.3ds');
         end;

      //black man
      10: begin
           mesh:=TAll3DSMesh.Create(nil);
           mesh.TexturePath:='data\characters\' + sChars[2] + '\';
           mesh.LoadFromFile('data\characters\' + sChars[2] + '\char.3ds');
         end;

      11: begin
           mesh:=TAll3DSMesh.Create(nil);
           mesh.TexturePath:='data\characters\' + sChars[3] + '\';
           mesh.LoadFromFile('data\characters\' + sChars[3] + '\char.3ds');
         end;
    end;
```

```
if ModelNr > 9 then
  terrorist := true
else
  terrorist := false;

//-------- set Spheres --------
//calc middle
xmax := mesh.Mesh[0].Maximum.x;
xmin := mesh.Mesh[0].Minimum.x;
ymax := mesh.Mesh[0].Maximum.y;
ymin := mesh.Mesh[0].Minimum.y;
zmax := mesh.Mesh[0].Maximum.z;
zmin := mesh.Mesh[0].Minimum.z;
for i := 0 to Length(mesh.FMesh) - 1 do
begin
  {for j := 0 to round((length(mesh.Mesh[i].FIndices)-1)/3) do
    for k := 0 to 2 do
      if ymax > mesh.Mesh[i].Vertex[Mesh.Mesh[i].Faces[j].Vertex[k]].y then
        ymax := mesh.Mesh[i].Vertex[Mesh.Mesh[i].Faces[j].Vertex[k]].y;
  //x   }
  if mesh.Mesh[i].Maximum.x > xmax then
    xmax := mesh.Mesh[i].Maximum.x;
  if mesh.Mesh[i].Minimum.x < xmin then
    xmin := mesh.Mesh[i].Minimum.x;
  //y
  if mesh.Mesh[i].Maximum.y > ymax then
    ymax := mesh.Mesh[i].Maximum.y;
  if mesh.Mesh[i].Minimum.y < ymin then
    ymin := mesh.Mesh[i].Minimum.y;
  //z
  if mesh.Mesh[i].Maximum.z > zmax then
    zmax := mesh.Mesh[i].Maximum.z;
  if mesh.Mesh[i].Minimum.z < zmin then
    zmin := mesh.Mesh[i].Minimum.z;
end;

//calc radius
Sphere.Center.y := (ymax+ymin)/2;
Sphere.radius   := (ymax-ymin)/2;

Sphere.Center.x       := 0;
Sphere.Center.z       := 0;

ChestSphere.radius     := 0.45*Sphere.radius;            //45% des Körpers
ChestSphere.Center.x   := 0;
ChestSphere.Center.y   := Sphere.Center.y + Sphere.radius*0.25;
ChestSphere.Center.z   := 0;

HeadSphere.radius      := 0.15*Sphere.radius;  //15% des Körpers
HeadSphere.Center.x    := 0;
HeadSphere.Center.y    := ChestSphere.Center.y + ChestSphere.radius + HeadSphere.radius;
```

```
    HeadSphere.Center.z    := 0;

    LegSphere.radius       := 0.4*Sphere.radius;             //40% des Körpers
    LegSphere.Center.x     := 0;
    LegSphere.Center.y     := ChestSphere.Center.y - ChestSphere.radius - LegSphere.radius;
    LegSphere.Center.z     := 0;

    ChestSphere.radius     := 0.5*Sphere.radius;  //50% -> eigentlich 45%, aber die Kugel soll
ein bisschen überlappen mit den beiden anderen.
    Move(position.x, HeadSphere.center.y-LegSphere.center.y-LegSphere.radius, position.z);

    Log.AddStatus('SphereRadius: ' + floattostr(Sphere.radius) +
                  'SphereY: ' + floattostr(Sphere.center.Y), 'UCharacters');

    //Create Displaylist...
    mesh.BuildDisplayList;        //remove - because of skelleton...
   end;
  end;
end;

procedure TCharacter.SetName(Value: string);
begin
  if _name <> Value then
  begin
    _name := Value;
    if Net.IsClient and (Index = Owner.ownChar) then
      Net.Client.hasInfos := true
    else
      if Net.IsServer then
      begin
        if Index > 0 then
          Net.Server.Clients[Index-1].hasInfos := true
        else
          Net.Server.hasInfos := true;
      end;
  end;
end;

procedure TCharacter.SetTerrorist(Value: boolean);
begin
  if _Terrorist <> Value then
  begin
    _Terrorist := Value;
    if Net.IsClient and (Index = Owner.ownChar) then
      Net.Client.hasInfos := true
    else
      if Net.IsServer then
      begin
        if Index > 0 then
          Net.Server.Clients[Index-1].hasInfos := true
        else
          Net.Server.hasInfos := true;
```

```
    end;
  end;
end;

procedure TCharacter.SetUsed(Value: boolean);
begin
  _Used := Value;
  if not Value then
    Dead  := true;
end;

procedure TCharacter.SetWeaponslot(Value: integer);
begin
  if _currentWeaponSlot <> Value then
  begin
    _currentWeaponSlot := Value;
    if Net.IsClient and (Index = Owner.ownChar) then
      Net.Client.hasInfos := true
    else
      if Net.IsServer then
      begin
        if Index > 0 then
          Net.Server.Clients[Index-1].hasInfos := true
        else
          Net.Server.hasInfos := true;
      end;
  end;
end;

{-------------------------------------------------------------------------------
 --------------------          TCharacters         -------------------------
 -------------------------------------------------------------------------------}
procedure TCharacters._SetLivingPlayers(Value: Integer);
begin
  _LivingPlayers := Value;
  if Value <= 1 then
  begin
    if numPlayers > 1 then
    begin
      WaitforPlayers := false;
      Respawn;
    end
    else
    begin
      if (LivingPlayers = 0) then
        Respawn;
      WaitforPlayers := true;
    end;
  end;
end;

procedure TCharacters._setOwnChar(Value: integer);
```

```
begin
  _OwnChar := Value;
  CurrentChar := char[Value];
end;

function TCharacters.AddChar(x,y,z: double; Model: integer): integer;
var
  i, j: integer;
begin
  j := -1;
  for i := 1 to Length(char) - 1 do
  begin
    if Char[i].used = false then
    begin
      j := i;
      Break;
    end;
  end;
  if j < 0 then
  begin
    setlength(Char, Length(Char) + 1);
    j := High(Char);
  end;

  char[j] := TCharacter.Create(j, Model, self);

  //inc(numBots);

  Char[j].Move(x,y,z);
  char[j].bot        := false;

  result:=j;
  inc(numPlayers);
  numChars := length(char);
end;

function TCharacters.CalcLivingPlayers: integer;
var
  i: integer;
begin
  numCTs := 0;
  numTerrors := 0;
  result := 0;
  for i := 0 to numChars - 1 do
  begin
    if Char[i].Terrorist then
      inc(numTerrors)
    else
      inc(numCts);

    if not Char[i].dead then
      inc(result);
```

```
  end;

  LivingPlayers := result;
end;

constructor TCharacters.Create;
begin
  inherited Create;
  numbots := 0;
  AddChar(0,0,0,-1);
  CurrentChar := Char[0];
  ownChar := 0;

  Quadric := GluNewQuadric;
  GluQuadricDrawStyle(Quadric, GLU_LINE);

  if Net.Client = nil then
    WaitforPlayers := true
  else
    WaitforPlayers := false;
end;

destructor TCharacters.Destroy;
var
  i: integer;
begin
  for i := 0 to High(char) do
    char[i].Free;
  inherited Destroy;
end;

procedure TCharacters.RearrangeCharArray;
var
  i,j: integer;
begin
  for i := 0 to Length(char)-1 do
    if char[i] = nil then
    begin
      for j := i to Length(char)-2 do          //letztes Element nicht miteinbeziehn, da dies im vorigen
Durchgang gemacht wurde.
        char[j] := char[j+1];
      setlength(char, Length(Char) - 1);
    end;
end;

procedure TCharacters.RemoveChar(_Char: integer);
begin
  if char[_Char].bot then
    dec(numbots);
  char[_Char] := nil;
  //RearrangeCharArray;          //Neu ordnen, um Geschwindigkeit zu gewinnen.
  //herausgenommen weil dies zu Fehlern führt.
```

```
end;

procedure TCharacters.RenderChars;
var
  i: integer;
begin
  for i := 0 to numChars - 1 do
    Char[i].Render;
end;

procedure TCharacters.Respawn;
var
  i: integer;
begin
  if (Net.isServer) and (not NewRespawn) then
  begin
    NextRespawn := Eventhandler.counter + 5000;
    NewRespawn  := true;
  end;
end;

end.
```

17. UKI

```
unit UKI;

interface

uses
  Basics,
  UCharacters;

type
  TKI = class(TObject)
    procedure Deleteallbots;
  end;

var
  KI: TKI;

implementation

uses
  Mainunit;

procedure TKI.Deleteallbots;
var
  i: integer;
begin
  for i:=0 to High(Chars.char) do
    if Chars.char[i].bot then
      Chars.RemoveChar(i);
```

```
end;


end.
```

18. ULevels

```
unit ULevels;

{
TODO:
 * PVS (potentially visible sets)
 * -> dafür wird occlusion culling benötigt, von der grafikkarte gegeben
 * Level sollten auch geladen werden können von hier
   * zusätzliche Informationen wie Fog, Licht usw?
 * Ein Algorithmus zum Bewegegen der Figuren, bool map für KI?


DONE:
 * Eine Strategie zur Lösung des Andauernden Texturwechseln?!        -> DONE, Materialien
   * Evtl. einfach über if schleifen abfragen ob die Textur in einem Vertex die gleiche ist, allerdings
ist dies leicht zeitraubend (bei ca 5k polygonen)
   * Oder das ganze wird irgendwie zu Texturen zusammengefügt
   * oder man versucht durch eine kluge Aufteilung in den Octrees nur in jedem Octree eine Textur
zu haben.
 * Wichtig: In diesem Format muss PVS abspeicherbar sein        DONE
 * Displaylisten?        DONE
 * 3ds sollte dann zu einem anderen Format konvertiert werden  DONE
 * Evtl werden durch 3ds Triangulation oder ähnliche Algorithmen gebraucht  DONE, wird nicht
gebraucht, 3ds speichert alles in Dreiecken
 * 3ds und .lvl format sollte abspeicherbar sein, darin sollte man versuchen über Streams
abzuspeichern   DONE
 * Materialien vollständig implementieren oder erst nur Texturen?   DONE
   * BumpMap?              //wird vorläufig nicht implementiert.
   * Diffuse, Ambient?     DONE
 * 3ds vollständig hinüberladen, Materialien.   DONE, siehe oben
 * Normalenberechnung // für jeden Punkt oder Dreieck?
 * Algorithmen müssen eingearbeitet werden, wie z.B. Line of sight -> UBasics


}

interface

uses
  UOctree,
  gl3ds,
  Filetypes,
  Dialogs,
  Basics,
  SysUtils,
  GUI,
  Windows,
```

```
  ULogger,
  dglOpenGL,
  IniFiles,
  Textures,
  UCharacters;

const
  VERSION = '0.2.5';

type
  PLevel = ^TLevel;
  TLevel = class(TObject)
    Octree: TOctree;
    name:   string;
    Skybox: Cardinal;
    TerSpawnPoints: array of TVector3f;
    CTSpawnPoints: array of TVector3f;
    procedure AddCTSpawnPoint(Position: TVector3f);
    procedure AddTerSpawnPoint(Position: TVector3f);
    constructor Create(_name: string; new: boolean = true);
    procedure Render;
    destructor Destroy; override;
    function  GetRespawnPos(Terrorist: boolean): TVector3f;
    function  GetTriangleData(Ray: TRayf): PPolygon;
    procedure LoadFromStream(B: TBinaryFile);
    procedure SaveToStream(B: TBinaryFile);
    procedure BuildSkybox(_name: string);
  end;

  procedure Convert3dstoLvl(name: string; _Max_Triangles_in_Node: integer);

  var
    SHOWNODES: boolean = false;

implementation

uses
  Mainunit, GUIAdds;

const
  light_position : Array[0..3] of GlFloat = (-100.0, 40.0, -100.0, 1.0);
  light_ambient  : Array[0..3] of GlFloat = (0.8, 0.8, 0.8, 1.0);
  light_diffuse  : Array[0..3] of GlFloat = (0.8, 0.8, 0.8, 1.0);

procedure Convert3dstoLvl(name: string; _Max_Triangles_in_Node: integer);
var
  Mesh: TAll3dsMesh;
  B: TBinaryFile;
  fname: string;
  i,j,k: integer;
  polys: array of PPolygon;
  Level: TLevel;
```

```
  counter, c2, c3: integer;
  temppoly: PPolygon;
begin
  fname := 'data\maps\'+name + '\map';
  Mesh := TAll3dsMesh.Create(nil);
  //mesh.TexturePath:= 'data\maps\'+name + '\';
  mesh.TexturePath := 'data\MapTextures\';
  mesh.LoadFromFile(fname+'.3ds');
  Level := TLevel.Create('', false);

  counter := 0;
  c2 := 0;
  c3 := 0;
  setlength(polys,0);

  for i := 0 to mesh.NumMeshes - 1 do
  begin
    counter := counter + ((length(mesh.Mesh[i].FIndices) - 1) div 3)+1;
  end;

  setlength(polys, counter);          //länge vorhersetzen, setlength ist extrem langsam und braucht
sauviel speicherplatz.
  counter := 0;

  if mesh.NumMeshes > 0 then
    for i := 0 to mesh.NumMeshes - 1 do
      with mesh.Mesh[i] do
        if mesh.Mesh[i].NumVertex > 0 then
        begin
          for j := 0 to ((length(mesh.Mesh[i].FIndices) - 1) div 3) do
          begin
            New(polys[counter]);
            for k := 0 to 2 do
            begin
              polys[counter].Vertexes[k].Vertex.x := Vertex[Faces[j].Vertex[k]].x;
              polys[counter].Vertexes[k].Vertex.y := Vertex[Faces[j].Vertex[k]].y;
              polys[counter].Vertexes[k].Vertex.z := Vertex[Faces[j].Vertex[k]].z;
              polys[counter].Vertexes[k].texCoord.x := Mapping[FIndices[j*3+k]].tu;
              polys[counter].Vertexes[k].texCoord.y := Mapping[FIndices[j*3+k]].tv;
            end;

            if FMatID <> nil then
            begin
              polys[counter].Material := mesh.Mesh[i].matID[j] + 1;    // + 1 weil es ein Material gibt
für nil.
              inc(c3);
            end
            else
            begin
              inc(c2);
              polys[counter].Material := 0;
            end;
```

```
          inc(counter);    //nummer des aktuellen polygons...
        end;
      end;

  for i := 0 to High(polys) do
  begin
    //CalculatePlane(polys[i]);
    CalculateNormals(polys[i]);
  end;

  Level.Octree := TOctree.Create(true, polys, _Max_Triangles_in_Node);

  Level.Octree.TexturePath := mesh.TexturePath;

  //Das Material 0 ist gleich dem nil Material, hat also keine Eigenschaften.
  setlength(Level.Octree.Materials, 1);
  Level.Octree.Materials[0] := TOctreeMaterial.Create(Level.Octree);
  if mesh.NumMaterials > 0 then
  begin
    setlength(Level.Octree.Materials, mesh.NumMaterials+1);   // ohne +1 weil es ein zusätzliches
Material gibt.
    for i := 0 to mesh.NumMaterials -1 do
    begin
      //Textur
      Level.Octree.Materials[i+1] := TOctreeMaterial.Create(Level.Octree);
      Level.Octree.Materials[i+1].hasTexture  := mesh.Material[i].HasTexturemap;
      Level.Octree.Materials[i+1].Texturename := mesh.Material[i].TextureFilename;

      //Materialien
      Level.Octree.Materials[i+1].HasSpecular := mesh.Material[i].IsSpecular;
      Level.Octree.Materials[i+1].HasDiffuse  := mesh.Material[i].IsDiffuse;
      Level.Octree.Materials[i+1].HasAmbient  := mesh.Material[i].IsAmbient;

      if mesh.Material[i].IsDiffuse then
      begin
        Level.Octree.Materials[i+1].Diffuse.r   := mesh.Material[i].DiffuseRed;
        Level.Octree.Materials[i+1].Diffuse.g   := mesh.Material[i].DiffuseGreen;
        Level.Octree.Materials[i+1].Diffuse.b   := mesh.Material[i].DiffuseBlue;
        Level.Octree.Materials[i+1].Diffuse.a   := mesh.Material[i].Transparency;
      end;

      if mesh.Material[i].IsSpecular then
      begin
        Level.Octree.Materials[i+1].Specular.r   := mesh.Material[i].SpecularRed;
        Level.Octree.Materials[i+1].Specular.g   := mesh.Material[i].SpecularGreen;
        Level.Octree.Materials[i+1].Specular.b   := mesh.Material[i].SpecularBlue;
        Level.Octree.Materials[i+1].Specular.a   := 1;
      end;

      if mesh.Material[i].IsAmbient then
      begin
```

```
      Level.Octree.Materials[i+1].Ambient.r   := mesh.Material[i].SpecularRed;
      Level.Octree.Materials[i+1].Ambient.g   := mesh.Material[i].SpecularGreen;
      Level.Octree.Materials[i+1].Ambient.b   := mesh.Material[i].SpecularBlue;
      Level.Octree.Materials[i+1].Ambient.a   := 1;
    end;

    //BumpMap
    Level.Octree.Materials[i+1].HasBumpmap := mesh.Material[i].HasBumpmap;
    if mesh.Material[i].HasBumpmap then
      Level.Octree.Materials[i+1].offset := mesh.Material[i].BumpmapStrength;

    //TwoSided
    Level.Octree.Materials[i+1].IsTwoSided := mesh.Material[i].TwoSided;
   end;
  end;

  B := TBinaryFile.Create(fname+'.lvl', true);
  Level.SaveToStream(B);
  Log.AddStatus(inttostr(counter) + ' Triangles in ' + inttostr(mesh.NumMeshes) + ' Meshes
found! With '+inttostr(c2)+' polys without Mats ('+inttostr(c3)+' with)', 'ULevels');

  Level.Destroy;
  B.CloseFile;
  Mesh.Destroy;
end;

procedure TLevel.AddCTSpawnPoint(Position: TVector3f);
var
  Ini: TIniFile;
  cur: integer;
begin
  setlength(CTSpawnPoints, Length(CTSpawnPoints) + 1);
  cur := High(CTSpawnPoints);
  CTSpawnPoints[cur] := Position;

  try
    Ini := TIniFile.Create('data\maps\' + name + '\config.ini');
    Ini.WriteInteger('config', 'CTSpawnPoints', Length(CTSpawnPoints));
    Ini.WriteFloat('CTSpawn' +  inttostr(cur), 'x', Position.x);
    Ini.WriteFloat('CTSpawn' +  inttostr(cur), 'y', Position.y);
    Ini.WriteFloat('CTSpawn' +  inttostr(cur), 'z', Position.z);
  finally
    Ini.Free;
  end;
end;

procedure TLevel.AddTerSpawnPoint(Position: TVector3f);
var
  Ini: TIniFile;
  cur: integer;
begin
  setlength(TerSpawnPoints, Length(TerSpawnPoints) + 1);
```

```
  cur := High(TerSpawnPoints);
  TerSpawnPoints[cur] := Position;

  try
    Ini := TIniFile.Create('data\maps\' + name + '\config.ini');
    Ini.WriteInteger('config', 'TerSpawnPoints', Length(TerSpawnPoints));
    Ini.WriteFloat('TerSpawn' +  inttostr(cur), 'x', Position.x);
    Ini.WriteFloat('TerSpawn' +  inttostr(cur), 'y', Position.y);
    Ini.WriteFloat('TerSpawn' +  inttostr(cur), 'z', Position.z);
  finally
    Ini.Free;
  end;
end;

constructor TLevel.Create(_name: string; new: boolean = true);
var
  polys: array of PPolygon;
  B: TBinaryFile;
  fname: string;
  Ini: TInifile;
  Skyboxname: String;
  num,i: integer;
begin
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT,  @light_ambient[0]);
  glLightfv(GL_LIGHT0, GL_DIFFUSE,  @light_diffuse[0]);
  glLightfv(GL_LIGHT0, GL_POSITION, @light_position[0]);
  //glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.0);
  //glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 2);
  //glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0);

  if new then
  begin
    name := _name;
    fname := 'data\maps\'+name + '\';
    B := TBinaryFile.Create(fname+'map.lvl', false);
    Octree := TOctree.Create(false, polys);
    Octree.name := _name+'map.lvl';

    //load config.ini
    try
      Ini := TIniFile.Create(fname+'config.ini');
      Skyboxname := Ini.ReadString('config', 'Skybox', 'default');
      BuildSkyBox(Skyboxname);

      num := Ini.ReadInteger('config', 'TerSpawnPoints', 0);
      for i := 0 to num - 1 do
      begin
        setlength(TerSpawnPoints, i + 1);
        TerSpawnPoints[i].x := Ini.ReadFloat('TerSpawn' +  inttostr(i), 'x', 0);
        TerSpawnPoints[i].y := Ini.ReadFloat('TerSpawn' +  inttostr(i), 'y', 0);
```

```
      TerSpawnPoints[i].z := Ini.ReadFloat('TerSpawn' +  inttostr(i), 'z', 0);
    end;

    num := Ini.ReadInteger('config', 'CTSpawnPoints', 0);
    for i := 0 to num - 1 do
    begin
      setlength(CTSpawnPoints, i + 1);
      CTSpawnPoints[i].x := Ini.ReadFloat('CTSpawn' +  inttostr(i), 'x', 0);
      CTSpawnPoints[i].y := Ini.ReadFloat('CTSpawn' +  inttostr(i), 'y', 0);
      CTSpawnPoints[i].z := Ini.ReadFloat('CTSpawn' +  inttostr(i), 'z', 0);
    end;

  finally
    Ini.Free;
  end;

  //load map
  LoadFromStream(B);
 end;
end;

procedure TLevel.Render;
begin
 //glEnable(GL_MULTISAMPLE_ARB);  //AntiAliasing
 with Chars.CurrentChar.position do
  gltranslatef(x,y,z);
 glCallList(Skybox);
 glLoadIdentity;


 //glUseProgramObjectARB(pobject);
 //glEnable(GL_LIGHTING);
 Octree.drawOctree(true,SHOWNODES);
 glDisable(GL_LIGHTING);
 //glUseProgramObjectARB(0);
 //glDisable(GL_MULTISAMPLE_ARB); //AntiAliasing
end;

destructor TLevel.Destroy;
begin
 Octree.Destroy;
 inherited Destroy;
end;

function  TLevel.GetRespawnPos(Terrorist: boolean): TVector3f;
begin
 if Terrorist then
 begin
  if Length(TerSpawnPoints) > 0 then
   result := TerSpawnPoints[  random(Length(TerSpawnPoints))  ]
  else
   result := vec3f(0,1,0);
```

```
  end
  else
  begin
    if Length(CTSpawnPoints) > 0 then
      result := CTSpawnPoints[  random(Length(CTSpawnPoints))  ]
    else
      result := vec3f(0,1,0);
  end;
end;

function  TLevel.GetTriangleData(Ray: TRayf): PPolygon;
var
  i,j: integer;
  dist: single;
  lastdist: single;
begin
  with Octree do
  begin
    result := nil;
    for i := 0 to Length(Octree.Nodes) - 1 do
      for j := 0 to Octree.Nodes[i].numv - 1 do
      begin
        if Col_Triangle_Ray_Return_Dist(Octree.Nodes[i].poly[j], Ray, Dist) then
        begin
          if result = nil then
          begin
            result := Octree.Nodes[i].poly[j];
            lastdist := Dist;
          end
          else
            if (lastDist > Dist) and (Dist > 0) then
            begin
              result := Octree.Nodes[i].poly[j];
              lastDist := Dist;
            end;
        end;
      end;
  end;
end;

procedure TLevel.LoadFromStream(B: TBinaryFile);
var
  Ini: TIniFile;
  fname: string;
begin
  fname := 'data\maps\'+name+'\';
  B.ReadHeader;
  if B.version <> ULevels.VERSION then
  begin
    B.CloseFile;
    Convert3dstoLvl(name, 300);
    B := TBinaryFile.Create(fname+'map.lvl', false);
```

```
    B.ReadHeader;
    LoadFromStream(B);
    B.CloseFile;
    Exit;
  end;
  Octree.LoadFromStream(B);
  B.CloseFile;

  Log.AddStatus('Loaded Level ' + name, 'ULevels');
end;

procedure TLevel.SaveToStream(B: TBinaryFile);
begin
  B.WriteHeader(ULevels.VERSION, 'Level');
  Octree.SaveToStream(B);
end;

procedure TLevel.BuildSkybox(_name: string);
const
  SkyBoxName: array[0..5] of String = ('pos_x', 'neg_x', 'pos_z', 'neg_z', 'pos_y', 'neg_y');
var
  i, d, j : integer;
  SkyboxTex: array[0..5] of Cardinal;
begin
  for i:=0 to 5 do
    Loadtexture('data\skyboxes\'+ _name+'\'+ SkyBoxName[i] + '.bmp',SkyboxTex[i], false);

  d:=300;
  j:= 1;
  Skybox := glGenLists(1);
  glNewList(Skybox, GL_COMPILE);
     glbindtexture(GL_TEXTURE_2D, SkyboxTex[0]);
     glBegin(GL_QUADS);
      glTexCoord2f(0, 0); glVertex3f(d-j,    -d,    d);
      glTexCoord2f(0, 1); glVertex3f(d-j,    d,    d);
      glTexCoord2f(1, 1); glVertex3f(d-j,    d,    -d);
      glTexCoord2f(1, 0); glVertex3f(d-j,    -d,    -d);
     glEnd;
     glbindtexture(GL_TEXTURE_2D, SkyboxTex[1]);
     glBegin(GL_QUADS);
      glTexCoord2f(0, 0); glVertex3f(-d+j,    -d,    -d);
      glTexCoord2f(0, 1); glVertex3f(-d+j,    d,    -d);
      glTexCoord2f(1, 1); glVertex3f(-d+j,    d,    d);
      glTexCoord2f(1, 0); glVertex3f(-d+j,    -d,    d);
     glEnd;
     glbindtexture(GL_TEXTURE_2D, SkyboxTex[2]);
     glBegin(GL_QUADS);
      glTexCoord2f(0, 0); glVertex3f(-d,    -d,    d-j);
      glTexCoord2f(0, 1); glVertex3f(-d,    d,    d-j);
      glTexCoord2f(1, 1); glVertex3f(d,    d,    d-j);
      glTexCoord2f(1, 0); glVertex3f(d,    -d,    d-j);
     glEnd;
```

```
    glbindtexture(GL_TEXTURE_2D, SkyboxTex[3]);
    glBegin(GL_QUADS);
     glTexCoord2f(0, 0); glVertex3f(d,   -d,   -d+j);
     glTexCoord2f(0, 1); glVertex3f(d,   d,   -d+j);
     glTexCoord2f(1, 1); glVertex3f(-d,   d,   -d+j);
     glTexCoord2f(1, 0); glVertex3f(-d,   -d,   -d+j);
    glEnd;
    glbindtexture(GL_TEXTURE_2D, SkyboxTex[4]);
    glBegin(GL_QUADS);
     glTexCoord2f(0, 0); glVertex3f(-d,   d-j,   -d);
     glTexCoord2f(0, 1); glVertex3f(d,   d-j,   -d);
     glTexCoord2f(1, 1); glVertex3f(d,   d-j,   d);
     glTexCoord2f(1, 0); glVertex3f(-d,   d-j,   d);
    glEnd;
    glbindtexture(GL_TEXTURE_2D, SkyboxTex[5]);
    //keine gescheite reihenfolge weil der boden nicht erkennbar ist wie er liegen muss. /08.04.07:
HÄ?! naja lassen wir den Kommentar mal.
    glBegin(GL_QUADS);
     glTexCoord2f(1, 1); glVertex3f(d,   -d+j,   d);
     glTexCoord2f(0, 1); glVertex3f(d,   -d+j,   -d);
     glTexCoord2f(0, 0); glVertex3f(-d,   -d+j,   -d);
     glTexCoord2f(1, 0); glVertex3f(-d,   -d+j,   d);
    glEnd;
 glEndList;
end;


end.
```

19. ULogger

```
unit ULogger;

{
  orignially part of the JEDI project.
  but changed for Terrorist's Revenche.
}
interface

{$WEAKPACKAGEUNIT OFF}

uses
  Classes,
  SysUtils;

type
  TLogger = class(TObject)
    private
      TextFile: TextFile;
      _FilePath: string;
      _ApplicationName: string;
    public
      procedure AddError( ErrorMessage : string; Location : string );
```

```
    procedure AddWarning( WarningMessage : string; Location : string );
    procedure AddStatus( StatusMessage : string; Location : string );
    constructor Create;
    destructor Destroy; override;
    property FilePath: string read _FilePath;
    property ApplicationName: string read _ApplicationName;
  end;

var
  Log : TLogger;

implementation

procedure TLogger.AddError(ErrorMessage, Location: string);
var
  S : string;
begin
  S := '*** ERROR *** : @ ' + TimeToStr(Time) + ' MSG : ' + ErrorMessage + ' IN : ' + Location
+ #13#10;
  WriteLn(TextFile, S);
  Flush(TextFile);
end;

procedure TLogger.AddStatus(StatusMessage, Location: string);
var
  S : string;
begin
  S := 'STATUS INFO : @ ' + TimeToStr(Time) + ' MSG : ' + StatusMessage + ' IN : ' + Location +
#13#10;
  WriteLn(TextFile, S);
  Flush(TextFile);
end;

procedure TLogger.AddWarning(WarningMessage, Location: string);
var
  S : string;
begin
  S := '=== WARNING === : @ ' + TimeToStr(Time) + ' MSG : ' + WarningMessage + ' IN : ' +
Location + #13#10;
  WriteLn(TextFile, S);
  Flush(TextFile);
end;

constructor TLogger.Create;
begin
  _FilePath := ExtractFilePath(ParamStr(0));
  _ApplicationName := ExtractFileName( ParamStr(0) );
  AssignFile(TextFile, _FilePath + ChangeFileExt(_ApplicationName, '.log'));
  ReWrite(TextFile);
end;

destructor TLogger.Destroy;
```

```
begin
  CloseFile(TextFile);
  inherited;
end;

initialization
begin
  Log := TLogger.Create;
  Log.AddStatus( 'Starting Application', 'Initialization' );
end;

finalization
begin
  Log.AddStatus( 'Terminating Application', 'Finalization' );
  Log.Free;
  Log := nil;
end;

end.
```

20. UNetwork

```
unit UNetwork;

{
  Hilfe fürs Netzwerk, waren vor allem folgende Seiten:
    http://www.pascalgamedevelopment.com/viewarticle.php?a=49&p=1#article
    http://www.pascalgamedevelopment.com/viewarticle.php?a=70&p=1#article
    http://jcatki.no-ip.org/SDL_net/SDL_net.html
    http://wiki.delphigl.com/index.php/SDL_Net


//Die Nummerierung des Servers IST domminant!!!
//CharNr = NetNr + 1
//Server: OwnNr = 0
}


interface

uses
  SysUtils,
  Basics,
  ULogger,
  sdl,
  sdl_net;

const
  //Server
  nServer_GameIsFull        = $01;
  nServer_ScenarioInfo      = $02;
  nServer_ClosingServer     = $03;
  nServer_Sendinfos         = $04;
```

```
  nServer_Chat             = $05;
  nServer_Position         = $06;
  nServer_SendStartInfo    = $07;
  nServer_PlayerJoined     = $08;
  nServer_PlayerQuit       = $09;
  nServer_ArmorChange      = $10;
  nServer_HealthChange     = $11;
  nServer_AddGold          = $12;
  nServer_StatChange       = $13;

  // Client
  nClient_InGame           = $01;
  nClient_Chat             = $02;
  nClient_Sendinfos        = $03;
  nClient_Shoot            = $04;

type
  TNetwork = class;
  TServer = class;

  TTCPPackage = record
     MessageType:    Byte;
     StringA:        String[25];
     StringB:        String[25];
     StringC:        String[25];
     Shoot:          TRayf;
   end;

  PUDPPlayerPackage = ^TUDPPLayerPackage;
  TUDPPlayerPackage = record
    PlayerPosition: TPlayerPosition;
    health: byte;
    char: byte;
  end;

  TServerClient = class(TObject)
    hasInfos:      boolean;  //for timebased movement
    ingame:        boolean;
    TCPsocket:     PTCPSocket;
    Index:         integer;
    Owner:         TServer;
    PIP:           PIPAddress;   //the sdl_net ip format
    UDPIP:         TIPAddress;
    constructor Create(sock: PTCPSocket; _Server: TServer);
    destructor Destroy; override;
    procedure SendPosition(Pos: TVector3f);
    procedure SendStartInfo;
    procedure SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string);
    function Update: boolean;
  end;

  TServer = class(TObject)
```

```
  private
    function GetNumPlayers: integer;
  public
    Clients:        array of TServerClient;
    numClients:     integer;
    TCPSocket:      PTCPSocket;

    UDPSocket:      PUDPSocket;
    UDPPacket:      PUDPpacket;
    UDPPackContent: TUDPPlayerPackage;

    AllSockets:     PSDLNet_SocketSet;
    TCPPort:        integer;
    UDPPort:        integer;
    MaxPlayers:     integer;
    Owner:          TNetwork;
    hasInfos:       boolean;

    Name:           string;
    FriendlyFire:   boolean;
    GoldperRound:   integer;

    function AddClient(Socket: PTCPsocket): integer;
    constructor Create(sUDPPort, sTCPPort, _MaxPlayers, _Gold: integer; _FriendlyFire: boolean;
_Name: string; _Owner: TNetwork);
    destructor Destroy; override;
    procedure Chat(Sender: Integer; s: String; All, Team1: boolean);
    procedure SendInfosToAll(Char: Integer);
    procedure SendPlayerData; //wird extern angesteuert durch "timebased movement"
    procedure SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string); overload;
    procedure Update;

    property NumPlayers: integer read GetNumPlayers;
  end;

  TClient = class(TObject)
    ingame:         boolean;
    TCPSocket:      PTCPSocket;
    UDPSocket:      PUDPSocket;
    AllSockets:     PSDLNet_SocketSet;

    UDPPacket:      PUDPpacket;
    UDPPackContent: TUDPPlayerPackage;

    Owner:          TNetwork;
    TCPIP:          TIPAddress;   //the sdl_net ip format
    UDPIP:          TIPAddress;
    ServerIP:       string;
    TCPPort:        Integer;
    UDPPort:        Integer;
    hasInfos:       boolean;
    ServerName:     string;
```

```
    PlayerName:     string;
    constructor Create(cIP: String; cTCPPort, cUDPPort: Integer; _PlayerName: string; _Owner:
TNetwork);
    destructor Destroy; override;
    procedure SendChat(s: string; TeamOnly: boolean);
    procedure SendPlayerData;
    procedure SendInfos;
    procedure SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string);
    procedure Update;
  end;

  TNetwork = class(TObject)
    //Server
    IsServer:   Boolean;
    Server:     TServer;
    //Client
    IsClient:   Boolean;
    Client:     TClient;
    procedure Chat(s: String; TeamOnly: boolean);
    constructor Create;
    destructor Destroy; override;
    procedure EndLevel;
    procedure Error(ErrorMsg: String);
    procedure StartClient(IP: String; TCPPort, UDPPort: Integer; PlayerName: string);
    procedure StartServer(MaxPlayers, TCPPort, UDPPort, Gold: Integer; Friendlyfire: boolean;
ServerName: string);
    procedure SendPlayerData;
    procedure SetInfos(Value: boolean);
    procedure Update;

    property hasInfos: boolean write setInfos;
  end;

  procedure SendTCPPackage(Socket: PTCPSocket; PackageID: byte; StringA, StringB, StringC:
string);
  function Net_MessageTypeToStr(MessageType: byte): string;

var
  Net: TNetwork;

implementation

uses
  Mainunit,
  UCharacters,
  EventHandlerUnit,
  GUIAdds,        //only used for the TNetwork class (Error handling & reading port/ip)
  GUI;            //..

function Net_MessageTypeToStr(MessageType: byte): string;
begin
  if Net.IsServer then
```

```
  begin
    //Server
    case MessageType of
      $01: result := 'nServer_GameIsFull'       ;
      $02: result := 'nServer_ScenarioInfo'     ;
      $03: result := 'nServer_ClosingServer'    ;
      $04: result := 'nServer_Sendinfos'        ;
      $05: result := 'nServer_Chat'             ;
      $06: result := 'nServer_Position'         ;
      $07: result := 'nServer_SendStartInfo'    ;
      $08: result := 'nServer_PlayerJoined'     ;
      $09: result := 'nServer_PlayerQuit'       ;
      $10: result := 'nServer_ArmorChange'      ;
      $11: result := 'nServer_HealthChange'     ;
      $12: result := 'nServer_AddGold'          ;
    else
      result := inttostr(MessageType);
    end;
  end else
    begin
      // Client
      case MessageType of
        $01: result := 'nClient_InGame'         ;
        $02: result := 'nClient_Chat'           ;
        $03: result := 'nClient_Sendinfos'      ;
        $04: result := 'nClient_Shoot'          ;
      else
        result := inttostr(MessageType);
      end;
    end;

end;

function IPToString( ip : TIPAddress ) : string;
var
  ipaddr : UInt32;
begin
  ipaddr := SDL_Swap32( ip.host );
  // output the IP address nicely
  Result := format( '%d.%d.%d.%d', [ ipaddr shr 24, ( ipaddr shr 16 ) and $000000FF,
    ( ipaddr shr 8 ) and $000000FF, ipaddr and $000000FF ] );
end;

procedure SendData(s: string; DataOut: PUDPpacket);
begin
  DataOut.len := Length(s);
  StrPLCopy(PChar(DataOut.data), PChar(s), DataOut.Len);
  DataOut.channel := -1;
  SDLNet_UDP_Send(Net.Client.UDPSocket, -1, DataOut);
end;

procedure PrintData(DataIn: PUDPpacket);
```

```
var s: string;
begin
  s := StrPas(PChar(DataIn.data));
  log.AddStatus('Time From '+IPToString(DataIn.address) + ' is '+ s, 'UNetwork');
end;


procedure SendTCPPackage(Socket: PTCPSocket; PackageID: byte; StringA, StringB, StringC:
string);
var
  Package: TTCPPackage;
begin
  Package.MessageType := PackageID;
  Package.StringA     := StringA;
  Package.StringB     := StringB;
  Package.StringC     := StringC;

  PackCollector.AddPacket(Socket, Package, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(Socket, @Package, SizeOf(TTCPPackage));
end;

{--------------------------------------------------------------------------}
//                TServerClient
{--------------------------------------------------------------------------}

constructor TServerClient.Create(sock: PTCPSocket; _Server: TServer);
var
  str: string;
begin
  inherited Create;
  TCPSocket := sock;
  ingame := false;
  Owner  := _Server;

  PIP := @sock.remoteAddress;

  str :=  IPtostring(PIP^);
  if SDLNet_ResolveHost(UDPIP, PChar(str), Owner.UDPPort) = -1 then
  begin
    Owner.Owner.Error('Could not Resolve the UDP Host...');
    Exit;
  end;
  Log.AddStatus('ServerClient, UDPAddress: ' + str + ':' + inttostr(Owner.UDPPort), 'UNetwork');

  SDLNet_TCP_AddSocket(Owner.AllSockets, TCPSocket);
end;

destructor TServerClient.Destroy;
var
  i: integer;
begin
  for i := 0 to Length(Owner.Clients) - 1 do
```

```
    if Owner.Clients[i] <> nil then
      if Owner.Clients[i].Index <> Index then
        SendTCPPacket(nServer_PlayerQuit, inttostr(Index), '', '');

  Log.AddStatus('Client disconnected: ' + inttostr(Index), 'UNetwork');
  SDLNet_TCP_DelSocket(Owner.AllSockets, TCPSocket);
  SDLNet_TCP_Close(TCPSocket);
  if (length(Chars.char) > Index + 1) then
    if (Chars.char[Index+1] <> nil) then
      Chars.char[Index+1].used := false;
  Owner.Clients[Index] := nil;
  inherited Destroy;
end;

procedure TServerClient.SendPosition(Pos: TVector3f);
var
  Pack: TTCPPackage;
begin
  Pack.MessageType  := nServer_Position;
  Pack.Shoot.Origin := Pos;

  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
end;

procedure TServerClient.SendStartInfo;
var
  Pack: TTCPPackage;
begin
  Pack.MessageType := nServer_SendStartInfo;

  Pack.Shoot.Origin.X := length(chars.char);
  Pack.Shoot.Origin.Y := Index + 1;
  Pack.StringA        := Owner.Name;

  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
end;

procedure TServerClient.SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string);
var
  Pack: TTCPPackage;
begin
  Pack.MessageType  := PackageID;
  Pack.StringA      := StringA;
  Pack.StringB      := StringB;
  Pack.StringC      := StringC;

  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
end;
```

```
function TServerClient.Update: boolean;     //Error: false
var
  Package: TTCPPackage;
  i: integer;
begin
  result := true;
  if SDLNet_SocketReady(PSDLNet_GenericSocket(TCPSocket)) then
  begin
    if SDLNet_TCP_Recv(TCPSocket, @Package, SizeOf(TTCPPackage)) > 0 then
    begin
      case Package.MessageType of
        nClient_InGame:
         begin
           ingame := true;
           SendStartInfo;
           Owner.SendInfosToAll(0);
           for i := 0 to Length(Owner.Clients) - 1 do
             if i <> Index then
               Owner.SendInfosToAll(i+1);
         end;

        nClient_Chat:
         begin
           if round(Package.Shoot.Origin.x) = 1 then
             Owner.Chat(Index, Package.StringA+Package.StringB+Package.StringC, false, not
Chars.char[Index+1].Terrorist)
           else
             Owner.Chat(Index, Package.StringA+Package.StringB+Package.StringC, true , not
Chars.char[Index+1].Terrorist)
         end;

        nClient_Sendinfos:
         begin
           with Chars.Char[Index+1] do
           begin
             Name := Package.StringA;
             if Package.StringB = '1' then
               Terrorist := true
             else
               Terrorist := false;

             Armor             := round(Package.Shoot.Origin.X);
             ModelNr           := round(Package.Shoot.Origin.Y);

             currentweaponslot  := round(Package.Shoot.Direction.X);
             Pistol.Pistol      := round(Package.Shoot.Direction.Y);
             Rifle.Rifle        := round(Package.Shoot.Direction.Z);
           end;
         end;

        nClient_Shoot:
         with Chars.char[Index+1] do
```

```
        begin
          case currentweaponslot of
            0: Shoot(package.Shoot, Weapons.Rifles[Rifle.Rifle]);
            1: Shoot(package.Shoot, Weapons.Pistols[Pistol.Pistol]);
          end;
        end;
      end;
    end else
    begin
      Free;
      result := false;
      Exit;
    end;
  end;
end;

{--------------------------------------------------------------------------}
//                TServer
{--------------------------------------------------------------------------}

function TServer.AddClient(Socket: PTCPsocket): integer;
var
  i,j: integer;
begin
  j := -1;
  for i := 0 to Length(Clients) - 1 do
  begin
    if Clients[i] = nil then
    begin
      j := i;
      Break;
    end;
  end;
  if j < 0 then
  begin
    j := High(Clients) + 1;
    setlength(Clients, Length(Clients) + 1);
  end;
  numClients := Length(Clients);
  Clients[j] := TServerClient.Create(Socket, self);
  Clients[j].Index := j;
  result := j;
end;

procedure TServer.Chat(Sender: Integer; s: String; All, Team1: boolean);
var
  Pack: TTCPPackage;
  s2:   string;
  i:    integer;
begin
  s2             := '<>' + Chars.Char[Sender+1].Name + ': ' + s;
  Pack.MessageType := nServer_Chat;
```

```
  Pack.StringA     := copy(s2,  0, 25);
  if Length(s2) > 25 then
  begin
    Pack.StringB   := copy(s2, 25, 25);
    if Length(s2) > 50 then
      Pack.StringC := copy(s2, 50, 25);
  end;

  for i := 0 to NumClients - 1 do
    if All then
      if Clients[i] <> nil then
        PackCollector.AddPacket(Clients[i].TCPSocket, Pack, SizeOf(TTCPPackage))
        //SDLNet_TCP_Send(Clients[i].TCPSocket, @Pack, SizeOf(TTCPPackage))
    else
      if (Chars.char[i+1].Terrorist <> Team1) then
        if Clients[i] <> nil then
          PackCollector.AddPacket(Clients[i].TCPSocket, Pack, SizeOf(TTCPPackage));
          //SDLNet_TCP_Send(Clients[i].TCPSocket, @Pack, SizeOf(TTCPPackage));

  //Print for Own PC:
  if All then
    Console.AddString(s2)
  else
    if Chars.char[0].Terrorist <> Team1 then
      Console.AddString(s2)
end;

constructor TServer.Create(sUDPPort, sTCPPort, _MaxPlayers, _Gold: integer; _FriendlyFire:
boolean; _Name: string; _Owner: TNetwork);
var
  _ip: TIPAddress;
begin
  inherited Create;
  UDPPort := sUDPPort;
  TCPPort := sTCPPort;
  MaxPlayers := _MaxPlayers;
  Owner := _Owner;
  numClients := 0;
  Name := _Name;
  FriendlyFire := _FriendlyFire;
  GoldperRound := _Gold;

  if SDLNet_ResolveHost(_ip, nil, sTCPPort) = -1 then
  begin
    Owner.Error('SDLNet_ResolveHost: ');
    Exit;
  end;

  TCPSocket := SDLNet_TCP_Open(_ip);
  if TCPSocket = nil then
  begin
    Owner.Error('SDLNet_TCP_Open: ');
```

```
    Exit;
  end;

  UDPSocket := SDLNet_UDP_Open(sUDPPort);
  if UDPSocket = nil then
  begin
    Owner.Error('SDLNet_UDP_Open: ');
    Exit;
  end;

  UDPPacket := SDLNet_AllocPacket(SizeOf(TUDPPlayerPackage));
  UDPPacket.channel := -1;
  UDPPacket.data := @UDPPackContent;
  UDPPacket.len  := SizeOf(TUDPPlayerPackage);

  AllSockets := SDLNet_AllocSocketSet(MaxPlayers+2);  //TCP + UDP + TCP*Player
  SDLNet_TCP_AddSocket(AllSockets, TCPSocket);
  SDLNet_UDP_AddSocket(AllSockets, UDPSocket);

  Log.AddStatus('Server started at TCPport '+IntToStr(sTCPPort) + '; UDPport: ' +
Inttostr(sUDPPort),'UNetwork');
end;

destructor TServer.Destroy;
var
  i: integer;
begin
  for i := 0 to Length(Clients) - 1 do
    if Clients[i] <> nil then
      Clients[i].Free;

  SDLNet_FreeSocketSet(AllSockets);
  SDLNet_UDP_Close(UDPSocket);
  SDLNet_TCP_Close(TCPSocket);
  Log.AddStatus('Server closed', 'UNetwork');
  Owner.IsServer := false;
  inherited Destroy;
end;

function TServer.GetNumPlayers: integer;
var
  i: integer;
begin
  result := 0;
  for i := 0 to Length(Clients) - 1 do
    if Clients[i] <> nil then
      inc(result);
end;

procedure TServer.SendInfosToAll(Char: Integer);
var
  Pack: TTCPPackage;
```

```
  i: integer;
begin
  with Chars.Char[Char] do
  begin
    Pack.MessageType := nServer_SendInfos;
    Pack.StringA := Name;
    if Terrorist then
      Pack.StringB := '1'
    else
      Pack.StringB := '0';
    Pack.StringC   := inttostr(Char);

    Pack.Shoot.Origin.X :=  Armor;
    Pack.Shoot.Origin.Y :=  ModelNr;

    Pack.Shoot.Direction.X := currentweaponslot;
    Pack.Shoot.Direction.Y := Pistol.Pistol;
    Pack.Shoot.Direction.Z := Rifle.Rifle;
  end;
  for i := 0 to NumClients - 1 do
    if Clients[i] <> nil then
      if i+1 <> Char then
        SDLNet_TCP_Send(Clients[i].TCPsocket, @Pack, SizeOf(TTCPPackage));
end;

procedure TServer.SendPlayerData;   //wird extern angesteuert durch "timebased movement"
var
  i,j: integer;
begin
  for i := 0 to numClients - 1 do
    if Clients[i] <> nil then
      for j := 0 to length(Chars.char) - 1 do
        if Chars.Char[j] <> nil then
          if (i<>j-1) and (Clients[i].ingame) then
          begin
            Log.AddStatus('Sent an UDP Packet (Server): ' + inttostr(j), 'UNetwork');
            //UDPPacket.channel            := -1;
            //UDPPacket.len               := SizeOf(TUDPPackageContent);
            UDPPackContent.PlayerPosition := Chars.Char[j].Position;
            UDPPackContent.Health         := Chars.Char[j].Health;
            UDPPackContent.Char           := j;
            UDPPacket.address            := Clients[i].UDPIP;
            //UDPPacket.data := @UDPPackContent;
            SDLNet_UDP_Send(UDPSocket, -1, UDPPacket);
          end;
end;

procedure TServer.SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string);
var
  i: integer;
  Pack: TTCPPackage;
begin
```

```
  Pack.MessageType  := PackageID;
  Pack.StringA      := StringA;
  Pack.StringB      := StringB;
  Pack.StringC      := StringC;

  for i := 0 to numClients - 1 do
    if Clients[i] <> nil then
    begin
      PackCollector.AddPacket(Clients[i].TCPSocket, Pack, SizeOf(TTCPPackage));
      //SDLNet_TCP_Send(Clients[i].TCPSocket, @Pack, SizeOf(TTCPPackage));
    end;
end;

procedure TServer.Update;
var
  tempsocket: PTCPsocket;
  done: boolean;
  i,j: integer;
  counter: integer;
begin
  if SDLNet_CheckSockets(AllSockets, 0) > 0 then
  begin
    while SDLNet_SocketReady(PSDLNet_GenericSocket(TCPSocket)) do
    begin
      // accept a connection coming in on server_tcpsock
      tempsocket := nil;
      done := false;
      while not done do
      begin
        tempsocket := SDLNet_TCP_Accept(TCPSocket);
        if tempsocket <> nil then
        begin
          if MaxPlayers > (NumPlayers + 1) then
          begin
            i := AddClient(tempsocket);

            for j := 0 to Length(Clients) - 1 do
              if Clients[j] <> nil then
                if Clients[j].Index <> i then
                  Clients[j].SendTCPPacket(nServer_PlayerJoined, inttostr(i+1), '', '');

            Chars.AddChar(0,0,0,-1);
            Log.AddStatus('Peer Connected, with IP: ' +
SDLNet_ResolveIP(Clients[i].TCPSocket.remoteaddress) +', port: ' +
inttostr(Clients[i].TCPSocket.remoteaddress.port), 'UNetwork');
            SendTCPPackage(Clients[i].TCPsocket, nServer_ScenarioInfo, game.CurrentLevel.name,
'', '');
          end else
          begin
            SendTCPPackage(tempsocket, nServer_GameIsFull, '', '', '');
            Console.AddStringAndLog('Game is full, one client has been kicked.', 'UNetwork');
            SDLNet_TCP_Close(tempsocket);
```

```
        end;
        tempsocket := nil;
      end else
      begin
        done := true;
      end;
    end;
  end;

  for i := 0 to numClients - 1 do
    if Clients[i] <> nil then
      if not Clients[i].Update then    //Bei Fehlern aus der Schleife raus.
        Exit;
 end;

 if SDLNet_SocketReady(PSDLNet_GenericSocket(UDPSocket)) then
 begin
  counter := 0;
  // This function can return 0 packets pending or -1 on error
  // so we want to check to see if we have more than 0 packets
  // pending.
  while (SDLNet_UDP_Recv(UDPSocket, UDPPacket) > 0) and (counter < 50) do
  begin
    inc(counter); //dass die Schleife nicht ewig weiterläuft und lags verursacht...

    // process the data etc
    UDPPackContent := PUDPPlayerPackage(UDPPacket.data)^;

    with Chars.char[UDPPackContent.char] do
    begin
      Move(UDPPackContent.PlayerPosition.x, UDPPackContent.PlayerPosition.y,
UDPPackContent.PlayerPosition.z);
      Position.Lookanglex := UDPPackContent.PlayerPosition.lookanglex;
      Position.Lookangley := UDPPackContent.PlayerPosition.lookangley;
    end;
    //PrintData(Owner.UDPPacket);
  {  Log.AddStatus('Received UDP Packet with length: ' + inttostr(UDPPacket.len) + '; from: ' +
                SDLNet_ResolveIP(UDPPacket.address) +', port: ' +
                inttostr(UDPPacket.address.port), 'UNetwork');
    Log.AddStatus('UDP: XYZ ' + inttostr(round(UDPPackContent.Playerposition.x)) + '|' +
                          inttostr(round(UDPPackContent.Playerposition.y)) + '|' +
                          inttostr(round(UDPPackContent.Playerposition.z)) + ' , health: ' +
                          inttostr(round(UDPPackContent.health)), 'UNetwork');       }
  end;
  //else
    //Log.AddError('UDP Disconnect!', 'UNetwork');
 end;
end;

{-------------------------------------------------------------------------------}
//               TClient
{-------------------------------------------------------------------------------}
```

```
constructor TClient.Create(cIP: String; cTCPPort, cUDPPort: Integer; _PlayerName: string;
_Owner: TNetwork);
var
  _ip: TIPAddress;
begin
  inherited Create;
  //set Pointers nil
  TCPSocket   := nil;
  UDPSocket   := nil;
  AllSockets  := nil;
  UDPPacket   := nil;

  TCPPort     := cTCPPort;
  UDPPort     := cUDPPort;
  ServerIP    := cIP;
  Owner       := _Owner;
  PlayerName  := _PlayerName;

  if SDLNet_ResolveHost(_ip,PChar(cIP),cTCPPort) = -1 then
  begin
    Owner.Error('SDLNet_ResolveHost: ');
    Free;
    Exit;
  end;

  TCPIP := _ip;

  TCPSocket := SDLNet_TCP_Open(_ip);

  if TCPSocket = nil then
  begin
    Owner.Error('SDLNet_TCP_Open: ');
    Free;
    Exit;
  end
  else
    Log.AddStatus('Client started at TCPport: '+IntToStr(cTCPPort)+ '; UDP Port:
'+IntToStr(cUDPPort)+'; with IP '+cIP,'UNetwork');

  UDPSocket := SDLNet_UDP_Open(cUDPPort);
  if UDPSocket = nil then
  begin
    Owner.Error('SDLNet_UDP_Open');
    Free;
    Exit;
  end
  else
    Log.AddStatus('Started UDP.','UNetwork');

  if SDLNet_ResolveHost(UDPIP, PChar(cIP), UDPPort) = -1 then
  begin
```

```
    Owner.Error('Could not Resolve the UDP Host...');
    Free;
    Exit;
  end;

  AllSockets := SDLNet_AllocSocketSet(2);      //UDP + TCP
  SDLNet_TCP_AddSocket(AllSockets, TCPSocket);
  SDLNet_UDP_AddSocket(AllSockets, UDPSocket);

  UDPPacket := SDLNet_AllocPacket(SizeOf(TUDPPlayerPackage));
  UDPPacket.channel := -1;
  UDPPacket.data := @UDPPackContent;
  UDPPacket.len  := SizeOf(TUDPPlayerPackage);

  Sleep(1000);  //Dem Client Zeit lassen.
end;

destructor TClient.Destroy;
begin
  SDLNet_FreePacket(UDPPacket);  //free packet

  SDLNet_UDP_Close(UDPSocket);  //free UDP Socket
  SDLNet_TCP_Close(TCPSocket);  //free TCP Socket

  SDLNet_FreeSocketSet(AllSockets);  //free the set

  Log.AddStatus('Client closed', 'UNetwork');
  Owner.IsClient := false;
  inherited Destroy;
end;

procedure TClient.SendChat(s: string; TeamOnly: boolean);
var
  Pack: TTCPPackage;
begin
  Pack.MessageType := nClient_Chat;
  Pack.StringA := '';
  Pack.StringB := copy(s, 0, 25);
  if length(s) > 25 then
    Pack.StringC := copy(s, 25, 25)
  else
    Pack.StringC := '';

  if TeamOnly then
    Pack.Shoot.Origin.x := 1
  else
    Pack.Shoot.Origin.x := 0;

  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
end;
```

```
procedure TClient.SendInfos;
var
  Pack: TTCPPackage;
begin
  with Chars.CurrentChar do
  begin
    Pack.MessageType := nClient_SendInfos;
    Pack.StringA := Name;
    if Terrorist then
      Pack.StringB := '1'
    else
      Pack.StringB := '0';

    Pack.Shoot.Origin.X :=  Armor;
    Pack.Shoot.Origin.Y :=  ModelNr;

    Pack.Shoot.Direction.X := currentweaponslot;
    Pack.Shoot.Direction.Y := Pistol.Pistol;
    Pack.Shoot.Direction.Z := Rifle.Rifle;
  end;
  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
end;

procedure TClient.SendPlayerData;
begin
  if ingame then
    if not Chars.Char[Chars.ownChar].dead then
    begin
      UDPPacket.address            := UDPIP;
      UDPPackContent.PlayerPosition := Chars.Char[Chars.OwnChar].Position;
      //UDPPackContent.PlayerPosition := Chars.CurrentChar.Position;
      UDPPackContent.char          := Chars.ownChar;
      UDPPackContent.health        := 1;
      UDPPacket.data := @UDPPackContent;
      SDLNet_UDP_Send(UDPSocket, -1, UDPPacket);
      Log.AddStatus('Sent an UDP Packet (Client)', 'UNetwork');
    end;
end;

procedure TClient.SendTCPPacket(PackageID: byte; StringA, StringB, StringC: string);
var
  i: integer;
  Pack: TTCPPackage;
begin
  Pack.MessageType  := PackageID;
  Pack.StringA      := StringA;
  Pack.StringB      := StringB;
  Pack.StringC      := StringC;

  PackCollector.AddPacket(TCPSocket, Pack, SizeOf(TTCPPackage));
  //SDLNet_TCP_Send(TCPSocket, @Pack, SizeOf(TTCPPackage));
```

```
end;

procedure TClient.Update;
var
  package: TTCPPackage;
  i: integer;
begin
  if SDLNet_CheckSockets(AllSockets, 0) > 0 then
  begin
    if SDLNet_SocketReady(PSDLNet_GenericSocket(TCPSocket)) then
    begin
      if SDLNet_TCP_Recv(TCPSocket, @Package, SizeOf(TTCPPackage)) > 0 then
      begin
        case Package.MessageType of
          nServer_GameIsFull: Owner.Error('Client: Server is full');

          nServer_ScenarioInfo:
            begin
              Log.AddStatus('Client: SERVER LÄUFT! mit Scenario: ' + package.StringA, 'UNetwork');
              game.startlevel(package.StringA);  //server starten
              SendTCPPacket(nClient_Ingame, '', '', '');
            end;

          nServer_ClosingServer:  Owner.Error('Server Disconnected');

          nServer_Sendinfos:
            with Chars.char[strtoint(Package.StringC)] do
            begin
              if strtoint(package.stringc) = 1 then
                halt;
              Name := Package.StringA;
              if Package.StringB = '1' then
                Terrorist := true
              else
                Terrorist := false;

              Armor   := round(Package.Shoot.Origin.X);
              ModelNr := round(Package.Shoot.Origin.Y);

              currentweaponslot :=  round(Package.Shoot.Direction.X);
              Pistol.Pistol     :=  round(Package.Shoot.Direction.Y);
              Rifle.Rifle       :=  round(Package.Shoot.Direction.Z);
            end;

          nServer_Chat:
            begin
              Console.AddString(package.StringA+package.StringB+package.StringC);
            end;

          nServer_Position:
            begin
              Chars.char[Chars.OwnChar].Move(package.Shoot.Origin.X,
```

```
                                package.Shoot.Origin.Y,
                                package.Shoot.Origin.Z);
    end;

  nServer_SendStartInfo:
    begin
      Log.AddStatus('Client: received start infos!', 'UNetwork');
      while length(Chars.char) < package.Shoot.Origin.X do
        Chars.AddChar(0,0,0,-1);
      chars.ownChar := round(package.shoot.origin.y);
      ServerName := Package.StringA;
      Console.AddString('Welcome on the Server: "' + ServerName + '"!');
      Chars.char[Chars.ownChar].Name := PlayerName;
      ingame := true;
      hasInfos := true;
    end;

  nServer_PlayerJoined:
    begin
      Log.AddStatus('Client: Player Joined', 'UNetwork');
      if chars.AddChar(0,0,0,-1) <> strtoint(package.StringA) then
        owner.Error('The Server has an other char order than you have!');
    end;

  nServer_PlayerQuit:
    begin
      Log.AddStatus('Client: Player Left', 'UNetwork');
      i := strtoint(package.StringA);
      Chars.char[i].used := false;
    end;

  nServer_ArmorChange:
    begin
      Chars.char[chars.ownChar].Armor := strtoint(package.StringA);
    end;

  nServer_HealthChange:
    begin
      Chars.char[chars.ownChar].Health := strtoint(package.StringA);
    end;

  nServer_StatChange:
    begin
      i := strtoint(package.StringA);
      if Chars <> nil then
        if i < Chars.numChars then
        begin
          Chars.char[i].Kills := strtoint(package.StringB);
          Chars.char[i].Deaths := strtoint(package.StringC);
        end;
    end;
end;
```

```
      end else
      begin
        Owner.Error('Server disconnected!');
        Owner.EndLevel;
        Exit;
      end;
    end;
  end;

  while SDLNet_SocketReady(PSDLNet_GenericSocket(UDPSocket)) do
  begin
    // This function can return 0 packets pending or -1 on error
    // so we want to check to see if we have more than 0 packets
    // pending.
    while SDLNet_UDP_Recv(UDPSocket, UDPPacket) > 0 do
    begin
      // here we could update the local time or
      // process the data etc
      UDPPackContent := PUDPPlayerPackage(UDPPacket.data)^;
      with Chars.char[UDPPackContent.char] do
      begin
        Move(UDPPackContent.PlayerPosition.x, UDPPackContent.PlayerPosition.y,
UDPPackContent.PlayerPosition.z);
        Position.Lookanglex := UDPPackContent.PlayerPosition.lookanglex;
        Position.Lookangley := UDPPackContent.PlayerPosition.lookangley;
        Health   := UDPPackContent.health;
      end;

     { Log.AddStatus('Received UDP Packet with length: ' + inttostr(UDPPacket.len) + '; from: ' +
                SDLNet_ResolveIP(UDPPacket.address) +', port: ' +
                inttostr(UDPPacket.address.port), 'UNetwork');
      Log.AddStatus('UDP: XYZ ' + inttostr(round(UDPPackContent.Playerposition.x)) + '|' +
                           inttostr(round(UDPPackContent.Playerposition.y)) + '|' +
                           inttostr(round(UDPPackContent.Playerposition.z)) + ' , health: ' +
                           inttostr(round(UDPPackContent.health)), 'UNetwork');    }
    end;
  end;
end;

{-------------------------------------------------------------------------------}
//                TNetwork
{-------------------------------------------------------------------------------}

procedure TNetwork.Chat(s: String; TeamOnly: boolean);
begin
  if isServer then
    Server.Chat(-1, s, not TeamOnly, not Chars.char[0].Terrorist);
  if isClient then
    Client.SendChat(s, TeamOnly);
end;

constructor TNetwork.Create;
```

```
begin
  inherited Create;
  Server := nil;
  Client := nil;
end;

destructor TNetwork.Destroy;
begin
  Server.Free;
  Client.Free;
  inherited Destroy;
end;

procedure TNetwork.EndLevel;
begin
  if IsServer then
    Server.Free;
  if IsClient then
    Client.Free;
  IsClient := false;
  IsServer := false;
  Server := nil;
  Client := nil;
end;

procedure TNetwork.Error(ErrorMsg: String);
begin
  game.onError(ErrorMsg+SDL_GetError, 'UNetwork');
end;

procedure TNetwork.StartClient(IP: String; TCPPort, UDPPort: Integer; PlayerName: string);
begin
  IsClient      := true;
  Client        := TClient.Create(IP,TCPPort,UDPPort, PlayerName, self);
end;

procedure TNetwork.StartServer(MaxPlayers, TCPPort, UDPPort, Gold: Integer; Friendlyfire:
boolean; ServerName: string);
begin
  IsServer      := true;
  Server        := TServer.Create(TCPPort, UDPPort, MaxPlayers, Gold, Friendlyfire, ServerName,
self);
end;

procedure TNetwork.SendPlayerData;
begin
  if isServer then
    Server.SendPlayerData;
  if isClient then
    Client.SendPlayerData;
end;
```

```
procedure TNetwork.SetInfos(Value: boolean);
begin
  if IsServer then
  begin
    if Chars.ownChar > 0 then
      Net.Server.Clients[Chars.ownChar - 1].hasInfos := true
    else
      Net.Server.hasInfos := true;
  end
  else
    if IsClient then
      Client.hasInfos := true;

end;

procedure TNetwork.Update;
begin
  if IsServer then
    Server.Update;
  if IsClient then
    Client.Update;
end;

initialization
  Net := TNetwork.Create;

finalization
  Net.Free;

end.
```

21. Unit_SDL

```
unit Unit_SDL;

interface

uses
  windows,
  sdl,
  sdl_net,
  dglOpenGL,
  sysutils,
  EventHandlerUnit,
  ULogger;

type
  TSDL_UNIT = class
    procedure Init_SDL;
    function  glResizeWindow( width : integer; height : integer ) : Boolean;
    procedure glHandleEvents;
    procedure Quit_App;
    function glTimer( interval : UInt32; param : Pointer ) : UInt32;
```

```
    procedure Init;
  private
    { Private declarations }
  public
    { Public declarations }
  end;


const
  INVALID_MODULEHANDLE = 0;

var
  Windows_Caption:    shortstring   =       'Terrorist''s Revenge';

  Screen_bpp:         integer   = 32;

  FPSCount:           LongInt   = 1;
  Done:              integer   = 0;
  videoFlags:         integer;
  surface:           PSDL_Surface;
  h_DC:              HDC;                    // Device Context
  h_RC:              HGLRC;                   // OpenGL Rendering Context
  h_Wnd:             HWND;                    // Handle aufs Fenster

  GLHandle:          HINST;
  GLUHandle:          HINST;
  FPS:               integer;

implementation

uses Mainunit, UCharacters;

procedure TSDL_Unit.Init;
begin
    // Initalisierung
  Log.AddStatus('Startet Initializing of SDL', 'Unit_SDL');
  Init_SDL;
  InitOpenGL;
  ReadExtensions;

  //gibt schönere Animationen           IN SDL UNIT VERSCHIEBEN SPÄTER -> nicht nötig hier
  glEnable(GL_TEXTURE_2D);         // Aktiviert Texture Mapping
  glShadeModel(GL_SMOOTH);                    // Aktiviert weiches Shading
  glClearColor(0.0, 0.0, 0.0, 1.0);         // Bildschirm löschen (schwarz)
  glClearDepth(1.0);                          // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);                     // Aktiviert Depth Testing
  glDepthFunc(GL_LEQUAL);                      // Bestimmt den Typ des Depth Testing
  glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Qualitativ bessere Koordinaten
Interpolation
  glEnable(GL_BLEND);                         //blending aktivieren
  glBlendFunc(GL_SRC_ALPHA, GL_ONE);               //die blendfunktion einstellen
```

```
  // Setzen eines Timers für FPS-Berechnung
  SDL_Init(SDL_INIT_TIMER);
  SDL_AddTimer(1000, @TSDL_UNIT.gltimer, nil);

  // Anpassen der Fenstergröße
  EventHandler.Resize(screen_width, screen_height);
  glResizeWindow(screen_width, screen_height);
end;


{------------------------------------------------------------------------------}
{                          SDL Initialieren                                    }
{------------------------------------------------------------------------------}
procedure TSDL_UNIT.Init_SDL;
var
  videoinfo:    PSDL_VideoInfo;
begin
// Initalisieren vom Simple DirectMedia Layer
  if ( SDL_Init( SDL_INIT_VIDEO ) < 0 ) then
  begin
    messagebox(0,'Initalisierung von SDL schlug fehl: ', SDL_GetError, MB_OK);
    Log.AddError('Initalisierung von SDL schlug fehl: ','Unit_SDL');
    Quit_App;
  end;

// Information über Grafikkarte einholen
  videoInfo := SDL_GetVideoInfo;
  if ( videoInfo = nil ) then
  begin
    messagebox(0,'Grafikkarte ließ sich nicht abfragen: ', SDL_GetError, MB_OK);
    Log.AddError('Grafikkarte ließ sich nicht abfragen: ','Unit_SDL');
    Quit_App;
  end;

  // Flags für den SDL-Grafikmodus setzen
  videoFlags := SDL_OPENGL or                // OpenGL-Unterstützung aktivieren
                SDL_DOUBLEBUF or             // Double Buffering aktivieren
                SDL_HWPALETTE;               // Palette in Hardware speichern

  if ( videoInfo.hw_available <> 0 ) then
    videoFlags := videoFlags or SDL_HWSURFACE
  else
    videoFlags := videoFlags or SDL_SWSURFACE;

  // Wird hardware blitting unterstützt?
  if ( videoInfo.blit_hw <> 0 ) then videoFlags := videoFlags or SDL_HWACCEL;

  // Setzen der OpenGL-Attribute
  SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
```

```
  SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
  SDL_GL_SetAttribute( SDL_GL_STENCIL_SIZE, 8 );
  SDL_GL_SetAttribute( SDL_GL_MULTISAMPLEBUFFERS, 1);
  SDL_GL_SetAttribute( SDL_GL_MULTISAMPLESAMPLES, 6);

  // Fenstertitel festlegen
  SDL_WM_SetCaption(pChar( Windows_Caption + ' ['+IntToStr(Round(FPSCount))+' FPS]') , nil);
  videoflags := videoFlags or SDL_RESIZABLE {or SDL_FULLSCREEN{or SDL_NOFRAME};    //
Enable window resizing
  //videoflags := // Initalisierung der Surface
  surface := SDL_SetVideoMode( Screen_width, Screen_height, Screen_bpp, videoflags );
  if ( surface = nil ) then
  begin
    messagebox(0,'Erzeugen einer OpenGL-Zeichenfläche schlug fehl: ', SDL_GetError, MB_OK);
    Log.AddError('Erzeugen einer OpenGL-Zeichenfläche schlug fehl: ' + SDL_GetError,'Unit_SDL');
    Quit_App;
  end;

  //Netzwerk initialisieren:  (SDLNet)
  if (SDLNet_Init < 0) then
  begin
    Log.AddError('SDLNet_init failed:' + SDLNet_GetError, 'Unit_SDL');
    Quit_App;
  end;
  Log.AddStatus('SDLNet_Init was successful', 'Unit_SDL');
end;



{-------------------------------------------------------------------------------}
{            Stellt das Fenster auf die richtige Grösse                         }
{-------------------------------------------------------------------------------}
function TSDL_UNIT.glResizeWindow( width : integer; height : integer ) : Boolean;
begin
// Verhindern von "Division by Zero"
  if ( height = 0 ) then height := 1;

  // Viewport und Projektions-Matrix aktualisieren
  glViewport( 0, 0, width, height );

  glMatrixMode( GL_PROJECTION );
    glLoadIdentity;
    gluPerspective( 45.0, width / height, 0.1, 600 );
  glMatrixMode( GL_MODELVIEW );

  // Rücksetzen der World-Matrix
  glLoadIdentity;

  // Vorgang erfolgreich
  result := true;
end;
```

```
{------------------------------------------------------------------------------}
{                Event Handler - Mouse Down, Key Down                          }
{------------------------------------------------------------------------------}
procedure TSDL_UNIT.glHandleEvents;
var
  event: TSDL_Event;
begin;
    // Verarbeiten der Events
    while ( SDL_PollEvent( @event ) = 1 ) do
    begin
      case event.type_ of

        // Beenden der Applikation
        SDL_QUITEV :
        begin
          Done := -1;
        end;

        // Taste wurde gedrückt
        SDL_KEYDOWN :
        begin
          EventHandler.KeyDown(@event.key.keysym);
        end;

        SDL_KEYUP :
        begin
          EventHandler.KeyUp(@event.key.keysym);
        end;

        SDL_MouseMotion :
        begin
          EventHandler.MouseMotion(@event.motion);
        end;

        SDL_MOUSEBUTTONDOWN:
        begin
          EventHandler.MouseDown(@event.button);
        end;

        SDL_MOUSEBUTTONUP:
        begin
          EventHandler.MouseUp(@event.button);
        end;

        // Fenster-Größe hat sich verändert
        SDL_VIDEORESIZE :
        begin
          EventHandler.Resize(event.resize.w,event.resize.h);
          //surface := SDL_SetVideoMode( event.resize.w, event.resize.h, Screen_bpp, videoflags );

          if ( surface = nil ) then
          begin
```

```
          Messagebox(0,'Surface bei Größenänderung verloren: ', SDL_GetError, MB_OK);
          Log.AddError('Surface bei Größenänderung verloren: ' + SDL_GetError,'Unit_SDL');
          Quit_App;
        end;

        glResizeWindow( event.resize.w, event.resize.h );
      end;
    end;
  end;
end;

{------------------------------------------------------------------------------}
{                  Terminieren der SDL-Anwendung                               }
{------------------------------------------------------------------------------}
procedure TSDL_UNIT.Quit_App;
  procedure CloseOpenGL;
  begin
    if GLHandle <> INVALID_MODULEHANDLE then
    begin
      FreeLibrary(Cardinal(GLHandle));
      GLHandle := INVALID_MODULEHANDLE;
    end;

    if GLUHandle <> INVALID_MODULEHANDLE then
    begin
      FreeLibrary(Cardinal(GLUHandle));
      GLUHandle := INVALID_MODULEHANDLE;
    end;

//    ClearProcAddresses;
    //ClearExtensions;
  end;
begin;
  // Freigeben der Ressourcen
  SDLNet_Quit;
  CloseOpenGL;
  SDL_QUIT;
end;

{------------------------------------------------------------------------------}
{                    Timer für Frames pro Sekunde                              }
{------------------------------------------------------------------------------}
function TSDL_UNIT.glTimer( interval : UInt32; param : Pointer ) : UInt32;
begin;
    {if game.resttime > 0 then                    //hint
      begin
        dec(game.resttime);
        if game.Resttime mod 60 < 10 then
          menu.texte[menu.timetext].Text:= (inttostr(trunc(game.Resttime/60)) + ':0' +
inttostr(game.Resttime mod 60))
        else
          menu.texte[menu.timetext].Text:= (inttostr(trunc(game.Resttime/60)) + ':' +
```

```
inttostr(game.Resttime mod 60));
     end;   }


   (*SDL_WM_SetCaption(pChar( Windows_Caption + ' [' + IntToStr(FPSCount) + ' FPS]' +
   ' [x,z: ' + IntToStr(round(game.chars[0].position.x)) + ', ' +
IntToStr(round(game.chars[0].position.z)) + '] '
   {+ '[anglex,angley: ' + floatToStr(game.chars[0].position.angle) + ', ' +
floatToStr(game.camera.angley) + '] ' }
   + IntToStr(game.weapons.currentweaponslot) + ', ' + IntToStr(game.chars[1].health) + ', '+
floatToStr(game.debugvec.x) + ', '
   + floatToStr(game.debugvec.y) + ', ' + floatToStr(game.debugvec.z)
   ) , nil);    *)
   FPS := FPSCount;
   FPSCount := 0;
   Result :=1000;
end;

end.
```

22. UOctree

```
unit UOctree;

interface

uses
  Windows, Messages,
  dglOpenGL,
  Basics,
  glFrustum,
  Filetypes,
  Textures;

type
  TOctree = class;

  TNode = class(TObject)
    pos       : TVector3f;              //Mittelpunkt des Würfels
    size      : single;                 //Grösse des Würfels
    numv      : Integer;                //Anzahl Vektoren im Würfel
    numc      : Byte;                   //Anzahl der Children
    poly      : array of PPolygon;      //Die Polygone im Würfel.
    children  : array of TNode;         //Die Kind-Würfel
    smallest  : Boolean;                //ob es der schmalste Würfel ist.
    VisibleNodes : array of TNode;      //für PVS (Potentially Visible Sets)
    index     : Smallint;               //Index der nodes, für die Übersicht und das Streaming.
    parent    : TNode;
    list      : integer;
    owner     : TOctree;
    function HowManyPolygons(var polygons : array of PPolygon;Add:Boolean):Integer;
    function PolygonIn(p:PPolygon):Boolean;              //"PolygonIn" testet, ob ein Polygon im
Node ist und liefert dann TRUE zurück.
```

```
    Procedure Divide(var polygons : array of PPolygon; Sender: TOctree);   //"Divide" teilt den
Node in 8 kleine Würfel.
    procedure DrawPolygons;                    //Zeichnet alle Polygone in einem Octree.
    procedure Check(RenderwithDispList: boolean); //"Check" überprüft den Node auf Sichtbarkeit
und zeichnet dann seine Polygone.
    procedure Draw;                    //Draw zeichnet sich.
    constructor Create(_owner: TOctree);
    destructor destroy; override;              //löscht sich selbst.
    procedure LoadFromStream(B: TBinaryFile; Sender: TOctree);   //Lädt einen Octree...
    procedure SaveToStream(B: TBinaryFile);  //Speichert einen Octree
    procedure CalcPVS;                 //Berechnet PVS
    function SphereInNode(const Sphere: TSpheref): boolean;
  end;

  TOctreeMaterial = class(TObject)
    //kann man erweitern durch andere Eigenschaften wie z.B. Color, Materialeigenschaften & Rest
in gl3ds.
    hasTexture:   boolean;
    TextureID:    Cardinal;
    Texturename:  string;
    owner:        TOctree;

    //ob ein Material zwei Seiten hat.
    IsTwoSided:   boolean;

    // BumpMap - noch nicht implementiert.
    hasBumpMap:   boolean;
    offset:       single;

    // Material property
    hasDiffuse:   boolean;
    hasAmbient:   boolean;
    hasSpecular:  boolean;

    Ambient:      TColor4f;
    Diffuse:      TColor4f;
    Specular:     TColor4f;
    Emissive:     TColor4f;
    Shininess:    single;
    Transparency: single;

    Color:        TColor3f;
    procedure SaveToStream(B: TBinaryFile);
    procedure LoadFromStream(B: TBinaryFile);
    procedure Apply;                   // ändert die Eigenschaften auf die des aktuellen Materials.
    constructor Create(_Owner: TOctree);
    destructor Destroy; override;
  end;

  TOctree = class(TObject)
  public
    mainnode:             TNode;  //der oberste Würfel
```

```
    Nodes:                array of TNode;  //werden für PVS und Streaming benötigt, damit jedem
einen klaren Index zugewiesen werden kann.
    MAX_TRIANGLES_IN_NODE:  integer;
    Materials:            array of TOctreeMaterial;
    isDrawing:             boolean;
    CurrentMaterial:        TOctreeMaterial;
    hasList:              boolean;
    name:                 string;
    TexturePath:            string;
    procedure drawOctree(drawPol,drawNodes:Boolean);   //das Zeichnen des Octrees
    constructor Create(new: boolean; var polygons : array of PPolygon; _Max_Triangles_in_node:
integer = 500);
    procedure drawwithPVS(drawNodes: Boolean);
    procedure CalcPVS;
    destructor Destroy; override;
    procedure MakeDisplaylists;
    procedure LoadFromStream(B: TBinaryFile);
    procedure SaveToStream(B: TBinaryFile);
  end;

var
  USE_DISPLAY_LISTS: boolean = true;

implementation

//------------------------------------------------------------------------------
// TNode
//------------------------------------------------------------------------------
function TNode.PolygonIn(p:PPolygon):Boolean;
var
  i:Integer;
begin
  Result := false;
  for i := 0 to 2 do
    if not Result then
      with p.Vertexes[i].vertex do
      begin
        if(x >= pos.x - size) and
          (y >= pos.y - size) and
          (z >= pos.z - size) and
          (x <= pos.x + size) and
          (y <= pos.y + size) and
          (z <= pos.z + size) then Result := true;
      end;
end;

function TNode.HowManyPolygons(var polygons : array of PPolygon; Add:Boolean):Integer;
var
  i:Integer;
  counter: integer;
begin
 counter:=0;
```

```
 for i:=0 to High(polygons) do
   if polygons[i] <> nil then
     if PolygonIn(polygons[i])then
     begin
       Inc(counter);
       if Add then
       begin
         setLength(poly,counter);
         poly[counter-1]:=polygons[i];
         polygons[i] := nil;
       end;
     end;
 if add then
   numv := counter;
 Result:=counter;
end;

procedure TNode.Divide(var polygons : array of PPolygon; Sender: TOctree);
var
  TempNodes:array[0..7] of TNode;
  i,j: integer;
  counter: integer;
  polcount: integer;
begin
  for i:=0 to 7 do
  begin
    TempNodes[i] := TNode.Create(Sender);
    TempNodes[i].parent := self;
    TempNodes[i].size := size/2;
    TempNodes[i].smallest := false;
  end;

  numc := 0;
  TempNodes[0].pos.x:=pos.x-size/2;
  TempNodes[0].pos.y:=pos.y+size/2;
  TempNodes[0].pos.z:=pos.z-size/2;

  TempNodes[1].pos.x:=pos.x+size/2;
  TempNodes[1].pos.y:=pos.y+size/2;
  TempNodes[1].pos.z:=pos.z-size/2;

  TempNodes[2].pos.x:=pos.x+size/2;
  TempNodes[2].pos.y:=pos.y+size/2;
  TempNodes[2].pos.z:=pos.z+size/2;

  TempNodes[3].pos.x:=pos.x-size/2;
  TempNodes[3].pos.y:=pos.y+size/2;
  TempNodes[3].pos.z:=pos.z+size/2;

  TempNodes[4].pos.x:=pos.x-size/2;
  TempNodes[4].pos.y:=pos.y-size/2;
  TempNodes[4].pos.z:=pos.z-size/2;
```

```
  TempNodes[5].pos.x:=pos.x+size/2;
  TempNodes[5].pos.y:=pos.y-size/2;
  TempNodes[5].pos.z:=pos.z-size/2;

  TempNodes[6].pos.x:=pos.x+size/2;
  TempNodes[6].pos.y:=pos.y-size/2;
  TempNodes[6].pos.z:=pos.z+size/2;

  TempNodes[7].pos.x:=pos.x-size/2;
  TempNodes[7].pos.y:=pos.y-size/2;
  TempNodes[7].pos.z:=pos.z+size/2;

  //check if a polygon is in 2 or more nodes.
  for i := 0 to High(polygons) do
    if polygons[i] <> nil then
    begin
      counter := 0;
      for j := 0 to High(Tempnodes) do
        if TempNodes[j].PolygonIn(polygons[i]) then
          inc(Counter);
      if counter > 1 then
      begin
        inc(numv);
        setlength(poly, numv);
        poly[numv-1] := polygons[i];
        polygons[i] := nil;
      end;
    end;

  for i:=0 to 7 do
  begin
    polcount := TempNodes[i].HowManyPolygons(polygons, false);
    if polcount > 0 then
    begin
      inc(numc);
      setLength(children, numc);
      children[numc-1] := TempNodes[i];
      setlength(Sender.Nodes, High(Sender.Nodes)+2);
      Sender.Nodes[High(Sender.Nodes)] := TempNodes[i];
      TempNodes[i].index := High(Sender.Nodes);
      if polcount < Sender.MAX_TRIANGLES_IN_NODE then
      begin
        children[numc-1].smallest := true;
        children[numc-1].HowManyPolygons(polygons,true);
      end;
    end else
      TempNodes[i].Free;
  end;
  for i:=0 to numc-1 do
    if children[i].smallest=false then children[i].Divide(polygons, Owner);
end;
```

```
procedure TNode.drawPolygons;
var
  i,j:Integer;
begin
  for i := 0 to numv-1 do
  begin
    //set Material
    if owner.CurrentMaterial = nil then
    begin
      owner.materials[poly[i].material].Apply;
      owner.CurrentMaterial := owner.materials[poly[i].material];
    end
    else
      if owner.materials[poly[i].material] <> owner.CurrentMaterial then
      begin
        owner.materials[poly[i].material].Apply;
        owner.CurrentMaterial := owner.materials[poly[i].material];
      end;

    //draw Polygon
    for j := 0 to 2 do
      with poly[i].vertexes[j] do
      begin
        glNormal3fv(@normal);       //evtl rausnehmen?
        if owner.materials[poly[i].material].hasTexture then
        glTexCoord2fv(@texcoord);
        glVertex3fv(@vertex);
      end;
  end;
end;

procedure TNode.check(RenderwithDispList: boolean);
var
  i:Integer;
begin
  if Frustum.IsBoxWithin(
     pos.x,pos.y,pos.z,
     size,size,size)=true then
  begin
    if RenderwithDispList then
      glCallList(List)
    else
      drawPolygons;

    for i:=0 to numc-1 do
      if not smallest then children[i].check(RenderwithDispList);
  end;
end;

procedure TNode.draw;
var
```

```
  i:Integer;
begin
  if Frustum.IsBoxWithin(
     pos.x,pos.y,pos.z,
     size,size,size)=false then exit;
  with pos do
  begin
    glBegin(GL_LINES);
      glVertex3f(x-size,y-size,z-size);
      glVertex3f(x+size,y-size,z-size);
      glVertex3f(x-size,y+size,z-size);
      glVertex3f(x+size,y+size,z-size);
      glVertex3f(x-size,y-size,z+size);
      glVertex3f(x+size,y-size,z+size);
      glVertex3f(x-size,y+size,z+size);
      glVertex3f(x+size,y+size,z+size);

      glVertex3f(x+size,y+size,z+size);
      glVertex3f(x+size,y-size,z+size);
      glVertex3f(x+size,y+size,z-size);
      glVertex3f(x+size,y-size,z-size);
      glVertex3f(x-size,y+size,z+size);
      glVertex3f(x-size,y-size,z+size);
      glVertex3f(x-size,y+size,z-size);
      glVertex3f(x-size,y-size,z-size);

      glVertex3f(x+size,y+size,z+size);
      glVertex3f(x+size,y+size,z-size);
      glVertex3f(x+size,y-size,z+size);
      glVertex3f(x+size,y-size,z-size);
      glVertex3f(x-size,y-size,z+size);
      glVertex3f(x-size,y-size,z-size);
      glVertex3f(x-size,y+size,z+size);
      glVertex3f(x-size,y+size,z-size);
    glEnd;
  end;

  for i:=0 to Length(children)-1 do
    children[i].draw;
end;

constructor TNode.Create(_owner: TOctree);
begin
  Owner := _owner;
end;

destructor TNode.destroy;
var
  i:Integer;
begin
  for i := 0 to numv - 1 do
    Dispose(poly[i]);
```

```
  for i:=0 to Length(children)-1 do
    children[i].Destroy;
  setLength(children,0);
  inherited Destroy;
end;

procedure TNode.LoadFromStream(B: TBinaryFile; Sender: TOctree);
var
  i: integer;
  _h: word;
  s: smallint;
  l: Extended;
  temppoly: TPolygon;
begin
  B.S.Read(pos,SizeOf(TVector3f));
  B.Read(size);
  B.Read(numv);
  setlength(poly, numv);
  for i := 0 to numv - 1 do
  begin
    new(poly[i]);
    B.S.Read(temppoly,SizeOf(TPolygon));
    poly[i]^ := temppoly;
    l := poly[i].Vertexes[0].Vertex.x;
  end;
  B.S.Read(numc, SizeOf(numc));
  B.Read(smallest);

  B.Read(_h);
  setlength(Visiblenodes, _h);
  for i := 0 to _h - 1 do
  begin
    B.Read(s);
    VisibleNodes[i] := Sender.Nodes[s];
  end;

  B.Read(Index);
  B.Read(s);
  if s > -1 then
  begin
    Parent := Sender.Nodes[s];
    setlength(Parent.children, length(Parent.children) + 1);
    Parent.children[High(Parent.children)] := self;
  end
  else
    Parent := nil;
  Owner := Sender;
end;

procedure TNode.SaveToStream(B: TBinaryFile);
var
  i: integer;
```

```
  _h: word;
  s: smallint;
  temppoly: TPolygon;
begin
  B.S.Write(pos,SizeOf(TVector3f));
  B.Write(size);
  B.Write(numv);
  for i := 0 to numv - 1 do
  begin
    temppoly := poly[i]^;
    B.S.Write(temppoly,SizeOf(TPolygon));
  end;
  B.S.Write(numc, SizeOf(numc));
  B.Write(smallest);

  _h := length(Visiblenodes);
  B.Write(_h);
  for i := 0 to _h-1 do
    B.Write(VisibleNodes[i].index);

  B.Write(Index);
  s := -1;
  if parent <> nil then
    B.Write(parent.index)
  else
    B.S.Write(s, SizeOf(parent.index));
end;

procedure TNode.CalcPVS;                    //Berechnet PVS
begin
  // mit einer extension arbeiten, occlusion culling glaube ich. gibt ein Tutorial auf
wiki.delphigl.com
end;

function TNode.SphereInNode(const Sphere: TSpheref): boolean;
begin
//not excact, just an approximization
  if ((pos.x-size-Sphere.radius) < Sphere.center.x) and
     ((pos.y-size-Sphere.radius) < Sphere.center.y) and
     ((pos.z-size-Sphere.radius) < Sphere.center.z) and
     ((pos.x+size+Sphere.radius) > Sphere.center.x) and
     ((pos.y+size+Sphere.radius) > Sphere.center.y) and
     ((pos.z+size+Sphere.radius) > Sphere.center.z) then
    result := true
  else
    result := false;
end;

//----------------------------------------------------------------------------
// TOctree
//----------------------------------------------------------------------------
constructor TOctree.Create(new: boolean; var polygons : array of PPolygon;
```

```
_Max_Triangles_in_node: integer = 500);
var
  i,j: integer;
  maxSize,minSize: single;
begin
  inherited Create;
  if new then
  begin
    MAX_TRIANGLES_IN_NODE := _Max_Triangles_in_node;
    maxsize:=polygons[0].Vertexes[0].vertex.x;
    minsize:=polygons[0].Vertexes[0].vertex.x;
    for i:=0 to High(polygons) do
      for j:=0 to 2 do
        with polygons[i].Vertexes[j].vertex do
        begin
          if x > maxSize then maxSize:=x;
          if y > maxSize then maxSize:=y;
          if z > maxSize then maxSize:=z;

          if x < minSize then minSize:=x;
          if y < minSize then minSize:=y;
          if z < minSize then minSize:=z;
        end;
    mainnode := TNode.Create(self);
    setlength(Nodes, 1);
    Nodes[0] := Mainnode;
    MainNode.parent := nil;
    MainNode.index := 0;
    MainNode.pos.x := (maxSize+minSize)/2;
    MainNode.pos.y := (maxSize+minSize)/2;
    MainNode.pos.z := (maxSize+minSize)/2;
    MainNode.size  := (maxSize-minSize)/2;
    MainNode.smallest := false;
    MainNode.Divide(polygons, self);
  end
  else
  begin
    //evtl kommt hier noch was.
  end;
  CurrentMaterial := nil;
  isDrawing := false;
  setlength(Materials, 0);

  TexturePath := '';
end;

procedure TOctree.drawOctree(drawPol,drawNodes:Boolean);
begin
  //Frustum.Calculate;          //Now in Mainunit.
  glEnable(GL_CULL_FACE);
  glEnable(GL_TEXTURE_2D);
  glEnable(GL_BLEND);
```

```
  glColor4f(1,1,1,1);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  if drawPol then
  begin
    if not hasList then
      MakeDisplayLists;
    //glEnable(GL_TEXTURE_2D);
    //glEnable(GL_LIGHTING);
    CurrentMaterial := nil;
    isDrawing := false;
    MainNode.check(USE_DISPLAY_LISTS);          //zeichnet     //zurücksetzen auf true
    if isDrawing then
    begin
      glEnd;
      isDrawing := false;
    end;
  end;
  if drawNodes then
  begin
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_LIGHTING);
    glColor3f(1,1,0);
    MainNode.draw;
  end;
  glEnable(GL_CULL_FACE);
  glEnable(GL_TEXTURE_2D);
  glEnable(GL_BLEND);
  glColor4f(1,1,1,1);
end;

procedure TOctree.drawwithPVS(drawNodes: Boolean);
var
  i: integer;
  CurrentNode: TNode;
begin
  //herausfinden in welchem Würfel sich die Kamera befindet....
  //also CurrentNode bestimmen

  if not hasList then
    MakeDisplayLists;
  isDrawing := false;
  CurrentMaterial := nil;
  for i := 0 to High(CurrentNode.VisibleNodes) do
    with CurrentNode.VisibleNodes[i] do
    begin
      if Frustum.IsBoxWithin(
        pos.x,pos.y,pos.z,
        size,size,size)=true then
        DrawPolygons;
    end;
  glEnd;
  isDrawing := false;
```

```
  if drawNodes then
    drawOctree(false,true);
end;

procedure TOctree.CalcPVS;
begin
  //
end;

destructor TOctree.Destroy;
var
  i: integer;
begin
  MainNode.Destroy;
  for i := 0 to High(Materials) do
    Materials[i].Destroy;
  inherited Destroy;
end;

procedure TOctree.MakeDisplayLists;
var
  i: integer;
begin
  for i := 0 to High(Nodes) do
  begin
    Nodes[i].list := glGenLists(1);
    glNewList(Nodes[i].list, GL_COMPILE);
      isDrawing := false;
      CurrentMaterial := nil;
      Nodes[i].DrawPolygons;
      glEnd;
    glEndList;
  end;

  isDrawing := false;
  hasList := true;
end;

procedure TOctree.LoadFromStream(B: TBinaryFile);
var
  i: integer;
  w: word;
begin
  B.Read(TexturePath);
  B.Read(MAX_TRIANGLES_IN_NODE);
  B.Read(w);
  setlength(Nodes, w);
  for i := 0 to w-1 do
  begin
    Nodes[i] := TNode.Create(self);
    setlength(Nodes[i].children, 0);
```

```
    Nodes[i].LoadFromStream(B, self);
  end;

  B.Read(w);
  setlength(Materials, w);
  for i := 0 to w-1 do
  begin
    Materials[i] := TOctreeMaterial.Create(self);
    Materials[i].LoadFromStream(B);
  end;
  mainnode := Nodes[0];
end;

procedure TOctree.SaveToStream(B: TBinaryFile);
var
  i : integer;
  w : Word;
begin
  B.Write(TexturePath);
  B.Write(MAX_TRIANGLES_IN_NODE);
  w := Length(Nodes);
  B.Write(w);
  for i := 0 to w-1 do
    Nodes[i].SaveToStream(B);

  w := Length(Materials);
  B.Write(w);
  for i := 0 to w-1 do
    Materials[i].SaveToStream(B);
end;


// ---------------------------------------------------------------------
//    TOctreeMaterial
// ---------------------------------------------------------------------
procedure TOctreeMaterial.SaveToStream(B: TBinaryFile);
begin
  B.Write(hasTexture);
  B.Write(TextureName);

  B.Write(IsTwoSided);

  B.Write(hasBumpMap);
  B.Write(offset);

  B.Write(hasDiffuse);
  B.Write(hasAmbient);
  B.Write(hasSpecular);

  B.Write(Shininess);
  B.Write(Transparency);
```

```
  B.S.Write(Ambient, SizeOf(TColor4f));
  B.S.Write(Diffuse, SizeOf(TColor4f));
  B.S.Write(Specular, SizeOf(TColor4f));
  B.S.Write(Emissive, SizeOf(TColor4f));

  B.S.Write(Color, SizeOf(TColor3f));
end;

procedure TOctreeMaterial.LoadFromStream(B: TBinaryFile);
begin
  B.Read(hasTexture);
  B.Read(TextureName);
  if hasTexture then
    LoadTexture(owner.TexturePath + TextureName, TextureID, false);

  B.Read(IsTwoSided);

  B.Read(hasBumpMap);
  B.Read(offset);

  B.Read(hasDiffuse);
  B.Read(hasAmbient);
  B.Read(hasSpecular);

  B.Read(Shininess);
  B.Read(Transparency);

  B.S.Read(Ambient, SizeOf(TColor4f));
  B.S.Read(Diffuse, SizeOf(TColor4f));
  B.S.Read(Specular, SizeOf(TColor4f));
  B.S.Read(Emissive, SizeOf(TColor4f));

  B.S.Read(Color, SizeOf(TColor3f));
end;

procedure TOctreeMaterial.Apply;
//Diese procedure ist relativ stark performancelastig wenn man sie ohne Displaylisten laufen lässt.
//Sie ist auch auf möglichst wenige OpenGL Befehle ausgelegt, also optimal für Displaylisten.
begin
  //Beendet das aktuelle Zeichnen der Dreiecke.
  if owner.isDrawing then
    glEnd;

  //glColor3f(1,1,1); //nur vorübergehend.

  //Setzt die Materialien.
  if owner.CurrentMaterial = nil then
  begin
    if hasTexture then
    begin
      glEnable(GL_TEXTURE_2D);
      glBindTexture(GL_TEXTURE_2D, TextureID);
```

```
    end
    else
      glDisable(GL_TEXTURE_2D);

    if IsTwoSided then
      glDisable(GL_CULL_FACE)
    else
      glEnable(GL_CULL_FACE);

    glColor4f(Color.r,Color.g,Color.b,Transparency);

    glMaterialfv(GL_FRONT, gl_ambient, @ambient);
    glMaterialfv(GL_FRONT, gl_diffuse, @diffuse);
    glMaterialfv(GL_FRONT, gl_specular, @specular);
    glMaterialfv(GL_FRONT, gl_shininess, @Shininess);
    glMaterialfv(GL_FRONT, gl_emission, @emissive);
  end
  else
  begin
    if (owner.CurrentMaterial.ambient.r <> ambient.r) or (owner.CurrentMaterial.ambient.g <>
ambient.g) or
       (owner.CurrentMaterial.ambient.b <> ambient.b) or (owner.CurrentMaterial.ambient.a <>
ambient.a) then
      glMaterialfv(GL_FRONT, gl_ambient, @ambient);

    if (owner.CurrentMaterial.diffuse.r <> diffuse.r) or (owner.CurrentMaterial.diffuse.g <>
diffuse.g) or
       (owner.CurrentMaterial.diffuse.b <> diffuse.b) or (owner.CurrentMaterial.diffuse.a <>
diffuse.a) then
      glMaterialfv(GL_FRONT, gl_diffuse, @diffuse);

    if (owner.CurrentMaterial.specular.r <> specular.r) or (owner.CurrentMaterial.specular.g <>
specular.g) or
       (owner.CurrentMaterial.specular.b <> specular.b) or (owner.CurrentMaterial.specular.a <>
specular.a) then
      glMaterialfv(GL_FRONT, gl_specular, @specular);

    if owner.CurrentMaterial.Shininess <> Shininess then
      glMaterialfv(GL_FRONT, gl_shininess, @Shininess);

    if (owner.CurrentMaterial.emissive.r <> emissive.r) or (owner.CurrentMaterial.emissive.g <>
emissive.g) or
       (owner.CurrentMaterial.emissive.b <> emissive.b) or (owner.CurrentMaterial.emissive.a <>
emissive.a) then
      glMaterialfv(GL_FRONT, gl_emission, @emissive);

    if owner.CurrentMaterial.hasTexture <> hasTexture then
    begin
      if hasTexture then
      begin
        glEnable(GL_TEXTURE_2D);
      end
```

```
      else
        glDisable(GL_TEXTURE_2D);
    end;

    if hasTexture then
      glBindTexture(GL_TEXTURE_2D, TextureID);

    if owner.CurrentMaterial.IsTwoSided <> IsTwoSided then
    begin
      if IsTwoSided then
        glDisable(GL_CULL_FACE)
      else
        glEnable(GL_CULL_FACE);
    end;

    if (owner.CurrentMaterial.Transparency <> Transparency) or (owner.CurrentMaterial.Color.r
<> Color.r)
            or (owner.CurrentMaterial.Color.g <> Color.g) or (owner.CurrentMaterial.Color.b <>
Color.b) then
    begin
      glColor4f(Color.r,Color.g,Color.b,Transparency);
    end;
  end;

  //fängt das Zeichnen wieder an.
  glBegin(GL_TRIANGLES);
  Owner.isDrawing := true;
end;

constructor TOctreeMaterial.Create(_Owner: TOctree);
begin
  inherited Create;
  Owner := _Owner;

  hasTexture := false;
  TextureID := 0;
  TextureName := '';

  hasDiffuse  := false;
  hasAmbient  := false;
  hasSpecular := false;

  Diffuse.r   := 0.8;
  Diffuse.g   := 0.8;
  Diffuse.b   := 0.8;
  Diffuse.a   := 1;

  Ambient.r   := 0;
  Ambient.g   := 0;
  Ambient.b   := 0;
  Ambient.a   := 1;
```

```
  Specular.r  := 0;
  Specular.g  := 0;
  Specular.b  := 0;
  Specular.a  := 1;

  Emissive.r  := 0.0;
  Emissive.g  := 0.0;
  Emissive.b  := 0.0;
  Emissive.a  := 1.0;

  Shininess     := 1;
  Transparency  := 1;

  isTwoSided    := false;
  hasBumpMap    := false;

  Color.r     := 1;
  Color.g     := 1;
  Color.b     := 1;
end;

destructor TOctreeMaterial.Destroy;
begin
  if HasTexture then
    gldeletetextures(1, @TextureID); //lets clean up afterwards...
  inherited Destroy;
end;

end.
```

23. UScreenShot

```
unit UScreenShot;

interface

uses
  dglOpenGL,
  Windows,
  Graphics,
  JPEG,
  sysUtils;

type
  BITMAPFILEHEADER = packed record
    bfType: Word;
    bfSize: DWORD;
    bfReserved1: Word;
    bfReserved2: Word;
    bfOffBits: DWORD;
  end;

  BITMAPINFOHEADER = packed record
```

```
    biSize: DWORD;
    biWidth: Longint;
    biHeight: Longint;
    biPlanes: Word;
    biBitCount: Word;
    biCompression: DWORD;
    biSizeImage: DWORD;
    biXPelsPerMeter: Longint;
    biYPelsPerMeter: Longint;
    biClrUsed: DWORD;
    biClrImportant: DWORD;
  end;

  TTGAHEADER = packed record
    tfType : Byte;
    tfColorMapType : Byte;
    tfImageType : Byte;
    tfColorMapSpec : Array[0..4] of Byte;
    tfOrigX : Word; //Array [0..1] of Byte;
    tfOrigY : Word;
    tfWidth : Word;
    tfHeight : Word;
    tfBpp : Byte;
    tfImageDes : Byte;
  end;

  procedure ScreenShot(const Name: string);

implementation

procedure BmpToJpg(const Filename: String; Quality: TJPEGQualityRange=100);
var
  Bmp: TBitmap;
  Jpg: TJpegImage;
begin
  Bmp:=TBitmap.Create;
  Jpg:=TJpegImage.Create;
  try
    Bmp.LoadFromFile(Filename);
    Jpg.CompressionQuality:=Quality;
    Jpg.Assign(Bmp);
    Jpg.SaveToFile(ChangeFileExt(Filename, '.jpg' ));
  finally
    Jpg.Free;
    Bmp.Free;
  end;
end;

procedure ScreenShotBMP(const Name : string);
 var F : file;
     FileInfo: BITMAPINFOHEADER;
     FileHeader : BITMAPFILEHEADER;
```

```
    pPicData:Pointer;
    Viewport : array[0..3] of integer;
begin
 //Speicher für die Speicherung der Header-Informationen vorbereiten
 ZeroMemory(@FileHeader, SizeOf(BITMAPFILEHEADER));
 ZeroMemory(@FileInfo, SizeOf(BITMAPINFOHEADER));

 //Größe des Viewports abfragen --> Spätere Bildgrößenangaben
 glGetIntegerv(GL_VIEWPORT, @Viewport);

 //Initialisieren der Daten des Headers
 FileHeader.bfType := 19778; //$4D42 = 'BM'
 FileHeader.bfOffBits := SizeOf(BITMAPINFOHEADER)+SizeOf(BITMAPFILEHEADER);

 //Schreiben der Bitmap-Informationen
 FileInfo.biSize := SizeOf(BITMAPINFOHEADER);
 FileInfo.biWidth := Viewport[2];
 FileInfo.biHeight := Viewport[3];
 FileInfo.biPlanes := 1;
 FileInfo.biBitCount := 32;
 FileInfo.biSizeImage := FileInfo.biWidth*FileInfo.biHeight*(FileInfo.biBitCount div 8);

 //Größenangabe auch in den Header übernehmen
 FileHeader.bfSize := FileHeader.bfOffBits + FileInfo.biSizeImage;

 //Speicher für die Bilddaten reservieren
 GetMem(pPicData, FileInfo.biSizeImage);
 try
  //Bilddaten von OpenGL anfordern (siehe oben)
  glReadPixels(0, 0, Viewport[2], Viewport[3], GL_BGRA, GL_UNSIGNED_BYTE, pPicData);

  //Und den ganzen Müll in die Datei schieben ;-)
  //Moderne Leute nehmen dafür auch Streams ...
  AssignFile(f, name);
  Rewrite( f,1 );
  try
   BlockWrite(F, FileHeader, SizeOf(BITMAPFILEHEADER));
   BlockWrite(F, FileInfo, SizeOf(BITMAPINFOHEADER));
   BlockWrite(F, pPicData^, FileInfo.biSizeImage );
  finally
   CloseFile(f);
  end;
 finally
  //Und den angeforderten Speicher wieder freigeben ...
  FreeMem(pPicData, FileInfo.biSizeImage);
 end;
end;

procedure ScreenShotTGA(const Name : string);
var
  DataBuffer : array of Byte;
  f : file;
```

```
  tgaHeader : TTGAHEADER;
  width, height : integer;
  DataSize:Integer;
  viewport : Array[0..3] of integer;
begin
  //Viewport-Größe lesen
  glGetIntegerv(GL_VIEWPORT, @viewport);
  width := viewport[2];
  height := viewport[3];

  //Größe der Daten berechnen
  DataSize := Width * Height * 3;

  //Größe des Puffers festlegen (Speicher reservieren)
  SetLength(DataBuffer,DataSize);

  // TGA Kopf mit Daten füllen
  ZeroMemory(@tgaHeader, SizeOf(tgaHeader));
  tgaHeader.tfImageType := 2; // TGA_RGB = 2
  tgaHeader.tfWidth := Width;
  tgaHeader.tfHeight := Height;
  tgaHeader.tfBpp := 24;

  //Daten von OpenGL anfordern
  glReadPixels(0,0,Width, Height, GL_BGR, GL_UNSIGNED_BYTE, @DataBuffer[0]);

  //Datei erstellen
  AssignFile(f, Name);
  Rewrite( f,1 );
  try
    // TGA Kopf in die Datei reinschreiben
    BlockWrite(F, tgaHeader, SizeOf(tgaHeader));

    // Die eigentlichen Bilddaten in die Datei schreiben
    BlockWrite(f, DataBuffer[0], DataSize );
  finally
    CloseFile(f);
  end;
end;

procedure ScreenShotJPG(const Name : string);
var
  str: string;
begin
  str := ChangeFileExt(Name, '.bmp' );
  ScreenShotBMP(str);
  BmpToJpg(str);
  DeleteFile(str);
end;

procedure ScreenShot(const Name: string);
var
```

```
  str: string;
begin
  str := copy(Name, Length(Name) - 2, 3);
  if str = 'bmp' then
    ScreenShotBMP(Name)
  else
    if str = 'tga' then
      ScreenShotTGA(Name)
    else
      if str = 'jpg' then
        ScreenShotJPG(Name);
end;

end.
```

24. UShader

```
unit UShader;

interface

uses
  dglOpenGL,
  Classes,
  SysUtils,
  ULogger,
  Basics,
  GUI,
  Textures,
  Math,
  GUIAdds;

type
  TShader = class
  private
    _running: boolean;
    procedure setrunning(Value: boolean);
  public
    ProgramObject: GLHandleARB;
    constructor Create(vertexpath, fragmentpath: string);
    destructor Destroy; override;
    procedure Start;
    procedure Stop;
    property running: boolean read _running write setrunning;
  end;

  TRenderpass = class(TObject)
    Texture:   cardinal;
    TexWidth:  integer;
    TexHeight: integer;
    FacWidth:  single;
    FacHeight: single;
    Shader:    TShader;
```

```
    constructor Create(Width, Height: integer; vertexshader, fragmentshader: string);
    destructor Destroy; override;
    procedure GetScreen;
    procedure onScreenResize(NewWidth, NewHeight: integer; newTex: boolean);
    procedure Render;
  end;


procedure InitShaders;
function glSlang_GetInfoLog(glObject : GLHandleARB) : String;
function AddFullShader(var ProgramObject: GLhandleARB; vertexpath, fragmentpath: string):
boolean;
function LoadShaderfromFile(var ShaderObject: GLhandleARB; const ShaderType: GLenum; const
path: string): boolean;

var
  Renderpass1, Renderpass2, Renderpass3: TRenderpass;

implementation

uses
  Mainunit;

procedure InitShaders;
begin
  if GameVariabeln.VideoConfig.Shader then
  begin
    //AddFullShader(pobject, 'data\shaders\v_ccomic_102.txt', 'data\shaders\f_ccomic_102.txt');
    //AddFullShader(pobject, 'data\shaders\v_storybook.txt', 'data\shaders\f_storybook.txt');
    //AddFullShader(pobject, 'data\shaders\v_empty.txt', 'data\shaders\f_empty.txt');
    //glUniform1iARB(glGetUniformLocationARB(pobject, 'papertex'), 1);


 {   glActiveTexture(GL_TEXTURE1);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, outline);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, outline);    }


    Renderpass1 := TRenderpass.Create(Screen_Width, Screen_Height,
'data\shaders\v_blur_row.txt', 'data\shaders\f_blur_row.txt');
    Renderpass2 := TRenderpass.Create(Screen_Width, Screen_Height,
'data\shaders\v_blur_column.txt', 'data\shaders\f_blur_column.txt');
    Renderpass3 := TRenderpass.Create(Screen_Width, Screen_Height,
'data\shaders\v_bloom.txt', 'data\shaders\f_bloom.txt');

    Renderpass3.Shader.Start;
    glUniform1iARB(glGetUniformLocationARB(Renderpass3.Shader.ProgramObject,
PGLCharARB('normal')), 1);
    Renderpass3.Shader.Stop;
```

```
  end
  else
  begin
    Renderpass1 := TRenderpass.Create(Screen_Width, Screen_Height, '', '');
  end;

  LoadTexture('data\shaders\storybook.tga', outline, false);
end;

function glSlang_GetInfoLog(glObject : GLHandleARB) : String;
var
  blen,slen : GLInt;
  InfoLog   : PGLCharARB;
begin
  glGetObjectParameterivARB(glObject, GL_OBJECT_INFO_LOG_LENGTH_ARB , @blen);
  if blen > 1 then
  begin
    GetMem(InfoLog, blen*SizeOf(GLCharARB));
    glGetInfoLogARB(glObject, blen, slen, InfoLog);
    Result := PChar(InfoLog);
    Dispose(InfoLog);
  end;
end;

function AddFullShader(var ProgramObject: GLhandleARB; vertexpath, fragmentpath: string):
boolean;
var
  txt: TStringList;
  VertexShaderObject, FragmentShaderObject: GLhandleARB;
begin
  result := true;
  if Fileexists(vertexpath) and fileexists(fragmentpath) then
  begin
    //Create
    ProgramObject       := glCreateProgramObjectARB;

    if not LoadShaderfromFile(VertexShaderObject, GL_VERTEX_SHADER_ARB, vertexpath) then
      result := false;
    if not LoadShaderfromFile(FragmentShaderObject, GL_FRAGMENT_SHADER_ARB,
fragmentpath) then
      result := false;

    //Shader an Program anhängen
    glAttachObjectARB(ProgramObject, VertexShaderObject);
    glAttachObjectARB(ProgramObject, FragmentShaderObject);

    //Shader löschen, die zwei Objekte werden nicht mehr gebraucht.
    //glDeleteObjectARB(VertexShaderObject);
    //glDeleteObjectARB(FragmentShaderObject);

    //linken ?!
    glLinkProgramARB(ProgramObject);
```

```
    Log.AddWarning(glSlang_GetInfoLog(ProgramObject), 'UShader: '+ vertexpath + ' | ' +
fragmentpath);
  end else
  begin
    Log.AddError('filename did not match', 'UShader');
    result := false;
  end;
end;

function LoadShaderfromFile(var ShaderObject: GLhandleARB; const ShaderType: Cardinal; const
path: string): boolean;
var
  source: PChar;
  txt: TStringList;
  len: integer;
  compiled: integer;
  error: string;
begin
  txt := TStringList.Create;
  txt.LoadFromFile(path);

  source := PChar(txt.Text);
  len := length(txt.Text);

  ShaderObject := glCreateShaderObjectARB(ShaderType);
  glShaderSourceARB(ShaderObject, 1, @source, @len);
  glCompileShaderARB(ShaderObject);

  error := glSlang_GetInfoLog(ShaderObject);
  if error <> '' then
    game.onError(error,'UShader');

  glGetObjectParameterivARB(ShaderObject, GL_OBJECT_COMPILE_STATUS_ARB, @compiled);

  if compiled <> GL_TRUE then
  begin
    Log.AddError('Couldn''t compile: '+path, 'UShader');
    result := false;
  end
  else
    result := true;

  txt.Free;
end;

{------------------------------------------------------------------------------
      TRenderpass
------------------------------------------------------------------------------}
constructor TRenderpass.Create(Width, Height: integer; vertexshader, fragmentshader: string);
begin
  inherited Create;
  TexWidth := 0;
```

```
  TexHeight := 0;
  if (vertexshader <> '') and (fragmentshader <> '') then
    Shader := TShader.Create(vertexshader, fragmentshader);
  onScreenResize(Width, Height, true);
end;

destructor TRenderpass.Destroy;
begin
  glDeleteTextures(1, @Texture);
  inherited Destroy;
end;

procedure TRenderpass.onScreenResize(NewWidth, NewHeight: integer; newTex: boolean);
var
  oldw, oldh: integer;
begin
  oldw := TexWidth;
  oldh := TexHeight;
  TexWidth  := round(Power(2,Trunc(ln(NewWidth)/ln(2))+1));
  TexHeight := round(Power(2,Trunc(ln(NewHeight)/ln(2))+1));
  FacWidth  := NewWidth/TexWidth;
  FacHeight := NewHeight/TexHeight;
  if not (oldw = TexWidth) and not (oldh = TexHeight) then
  begin
    if not newTex then
      glDeleteTextures(1, @Texture);
    Texture := CreateEmptyTexture(TexWidth, TexHeight, 3);
  end;
  if Shader <> nil then
  begin
    Shader.Start;
    Console.AddStringAndLog('Screen Rezize: w/h/facw/fach: ' + floattostr(TexWidth) + '|' +
floattostr(TexHeight) + '|' + floattostr(FacWidth) + '|' + floattostr(FacHeight), 'UShader');
    glUniform1fARB(glGetUniformLocationARB(shader.ProgramObject, PGLCharARB('width')),
TexWidth);
    glUniform1fARB(glGetUniformLocationARB(shader.ProgramObject, PGLCharARB('height')),
TexHeight);
    glUniform1fARB(glGetUniformLocationARB(shader.ProgramObject, PGLCharARB('facwidth')),
FacWidth);
    glUniform1fARB(glGetUniformLocationARB(shader.ProgramObject, PGLCharARB('facheight')),
FacHeight);
    Shader.Stop;
  end;
end;

procedure TRenderpass.Render;
begin
  glBindTexture(GL_TEXTURE_2D, Texture);
  if Shader <> nil then
    glUseProgramObjectARB(Shader.ProgramObject);
  glRenderQuad(0,0,Screen_Width,Screen_Height,-35, 0,0,FacWidth,FacHeight);
  if Shader <> nil then
```

```
    glUseProgramObjectARB(0);
end;

procedure TRenderpass.GetScreen;
begin
  glBindTexture(GL_TEXTURE_2D, Texture); //Textur Renderpass binden
  glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 0, 0, TexWidth, TexHeight, 0); //Gerendertes
auf die Textur kopieren
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT); //wieder buffer löschen
end;

{------------------------------------------------------------------------------
       TShader
------------------------------------------------------------------------------}
constructor TShader.Create(vertexpath, fragmentpath: string);
begin
  inherited Create;
  AddFullShader(ProgramObject, vertexpath, fragmentpath);
end;

destructor TShader.Destroy;
begin
  //Shader löschen
  glDeleteObjectARB(ProgramObject);
  inherited Destroy;
end;

procedure TShader.setrunning(Value: boolean);
begin
  _running := Value;
  if Value then
    glUseProgramObjectARB(ProgramObject)
  else
    glUseProgramObjectARB(0);
end;

procedure TShader.Start;
begin
  running := true;
end;

procedure TShader.Stop;
begin
  running := false;
end;


end.
```