Praktyczny kurs programowania w Delphi na przykładzie wygaszacza ekranu Windows

Ryszard Paweł Kostecki

rpkost@fuw.edu.pl
www.rysieq.prv.pl

17 czerwca 2002*

Motto:

Nasz język, tak jak i nasza poezja, wywodzi się z arytmetycznego, binarnego funkcjonowania naszego mózgu. Klasyfikujemy na tak, na nie, na pozytywnie, na negatywnie. (...) Mój język dowodzi jedynie powolności wizji świata zredukowanej do binarnego punktu widzenia. Ta niewystarczalność języka jest oczywista i dokuczliwa. Ale cóż można powiedzieć o niewystarczalności inteligencji binarnej jako takiej? Wewnętrzna egzystencja, właściwa esencja spraw wymyka jej się. Może odkryć, że światło jest równocześnie ciągłe i nieciągłe, że cząsteczka benzenu ustala między swoimi sześcioma atomami wiązania podwójne, niemniej wzajemnie się wykluczające; ona uznaje to wszystko, ale nie pojmuje, jej własna struktura nie jest w stanie przyswoić sobie rzeczywistości złożonych struktur, które bada...

Louis Pauwels & Jacques Bergier, *Le matin des magiciens*

Streszczenie

Praca ta jest praktycznym kursem języka i środowiska programistycznego Delphi. Kurs ten polega na dokładnym omówieniu konstrukcji prostego wygaszacza ekranu napisanego w Delphi. W efektywnej lekturze tej pracy przydatna będzie znajomość jakiegoś języka programowania wyższego poziomu.

Abstract

This paper is a practical course of the Delphi language and programming environment. This course proceeds comprehensive discussion of simple screensaver written in Delphi. For the effecive reading of this paper, some knowledge in any high level programming language will be usefull.

---

*poprawki: 11 lutego 2003, 3 sierpnia 2003, 1 czerwca 2004, 13 listopada 2004.

Wstęp

Klasyczne programowanie polegało kiedyś na otworzeniu edytora i linia po linii wklepywaniu tego co ma komputer czynić. W tych zamierzchłych czasach pisano bardzo często w kodzie maszynowym lub w assemblerze. Był to język bardzo nieprzejrzysty i zależny ściśle od hardware'u (np. programy napisane na procesory Motoroli nie dawały się uruchomić na procesorach Intela i vice versa). Jednakże wzrost mocy obliczeniowej komputerów umożliwił powszechne pisanie programów w językach wysokiego poziomu, znacznie bardziej intuicyjnych, oraz niezależnych od platformy sprzętowej. Spośród rozmaitych języków programowania wysokiego poziomu najbardziej popularnymi i wszechstronnymi okazały się C/C++ i Pascal.

Współcześnie najbardziej popularnymi są obiektowo zorientowane języki C++, Delphi (następca Object Pascala), oraz Java. Programowanie polega dziś na zintegrowanym współdziałaniu różnych środowisk i standardów, zarówno programistycznych, jak i użytkownika. Jest to tzw. *integrated development*. Współcześnie programuje się głównie „składając z klocków" – łącząc ze sobą rozmaite moduły, klasy, obiekty, nie interesując się przy tym ich wewnętrznymi zachowaniami, a jedynie wejściem/wyjściem. Jest to tzw. OOP (*object oriented programming*). Programista nie zajmuje się tym, **jak** obiekty mają coś zrobić, ale **co** mają zrobić. Funkcjonowanie programu nie polega na realizacji kolejnych linii kodu, tylko na reakcjach obiektów na zdarzenia generowane przez inne obiekty. Jest to tzw. *event handling*. Powszechny stał się również wymóg łatwości modyfikacji, rozszerzalności, przenośności i ponownej używalności kodu (*extensibility, reusability*, ...). Delphi jest powszechnie użytkowanym środowiskiem programistycznym spełniającym te cechy. Kurs ten, po krótkim wstępie teoretycznym, będzie praktyczną nauką środowiska i języka Delphi na przykładzie konstrukcji wygaszacza ekranu.

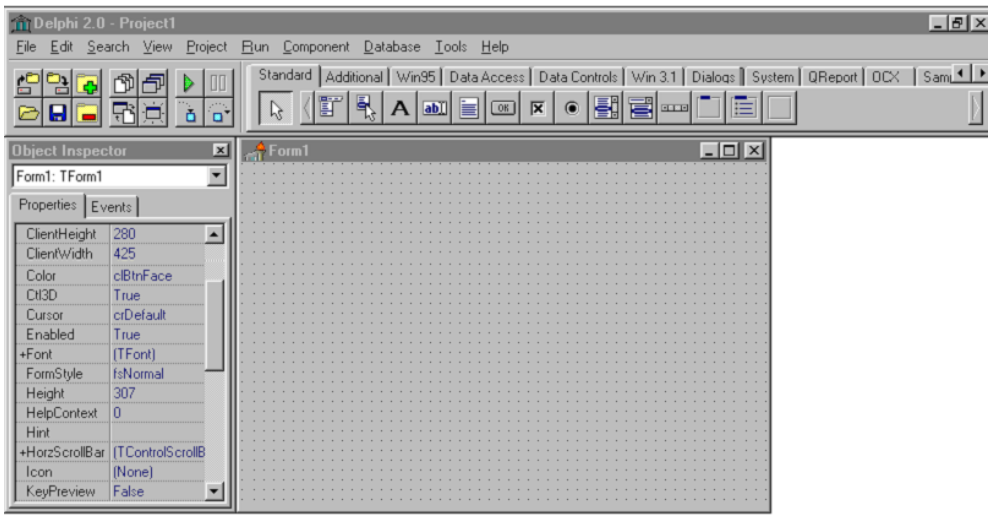
Spis treści

1 Programowanie w Delphi

1.1 VCL: klucz do programowania wizualnego i projektowania w stylu drag-and-drop

Rysunek 1: Paleta komponentów.

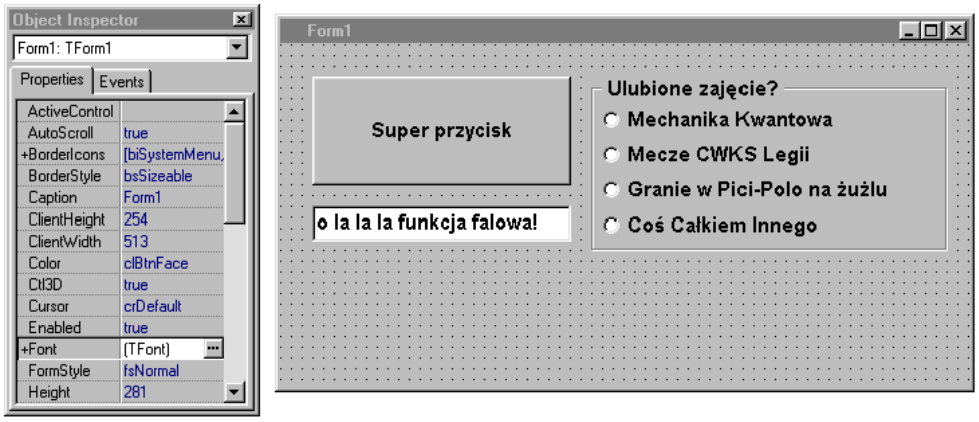
Programowanie w Delphi opiera się na bibliotece wizualnych komponentów VCL (Visual Component Library). Komponenty są to klocki, których programiści używają do stworzenia interfejsu użytkownika oraz oprogramowania niektórych spośród niewizualnych właściwości aplikacji. Dla tworzącego aplikację programisty komponent jest głównie obiektem, którego przeciąga się myszką z palety komponentów i umieszcza na formie (formą nazywa się programistyczny szkic okienka programu). Następnie, już po umieszczeniu komponentu na formie, można manipulować jego właściwościami oraz oprogramowywać specyfikę zachowań komponentu na rozmaite zdarzenia. Niektóre komponenty reagują na zdarzenia systemu (np. zegar czasu rzeczywistego TTimer) inne zaś pełnią rolę „naczyń pośredniczących" - wlewa się do nich pewne zdarzenia, a one generują inne... VCL zawiera w sobie komponenty w pełni obsługujące wszystkie elementy interfejsu użytkownika Windows: drzewa, paski narzędzi, menu, paski postępu, edytorki, itp., jak również wiele innych dodatkowych komponentów. Te dodatkowe komponenty uzupełniają istniejący interfejs użytkownika, obsługę danych, zdarzeń czy też narzędzi multimedialnych, wszystko w celu przyspieszenia procesu produkcji aplikacji. Tworzenie aplikacji zachodzi więc poprzez proste przeciąganie komponentów z palety komponentów (Visual Component Palette) na obszar projektowania okienka, oraz ustawianie ich właściwości w inspektorze obiektów (Object Inspector). Kod obsługi zdarzenia jest generowany automatycznie, oszczędzając czasu programiście, jakkolwiek ten ma pełny dostęp do niego i może go modyfikować wedle potrzeb.



Rysunek 2: Widok okien Delphi.

1.2 Teoria programowania w Delphi

Windows jest wielozadaniowym systemem okienkowym, w którym centralną rolę pełni reagowanie i zarządzanie zdarzeniami, generowanymi przez wiele procesów naraz. Dlatego też centralnym pojęciem w programowaniu



Rysunek 3: Komponowanie zabudowy okienka (formy).

Windowsowym jest właśnie zdarzenie. W ogólności program nie określa, jakie będą i mogą być następne czynności użytkownika. Program musi być przygotowany na rozmaite jego zachowania.

Programowanie w Delphi to głównie programowanie oparte na bibliotece VCL, czyli Visual Component Library. Biblioteka ta oparta jest na modelu interakcji pomiędzy właściwościami (ang. properties), metodami (ang. methods), oraz zdarzeniami (ang. events). Programowanie w VCL polega w dużym stopniu na kodowaniu wzajemnych interakcji pomiędzy komponentami: każdy z nich ma jakieś właściwości, metody, oraz zbiór reakcji na zdarzenia.

Najprostszym przykładem komponentu jest komponent TButton. Zawiera on w sobie całą obsługę tego, co użytkownik widzi jako przycisk. W tym: położenie przycisku w okienku, jego rozmiar, napis na nim (właściwości), czy też opis reakcji na kliknięcie na niego (metody/zdarzenia).

Jednym z podstawowych komponentów VCL jest komponent TForm, czyli tzw. forma. Forma jest komponentem obsługującym okienko windowsowe, wraz ze wszystkim co się na nim znajduje (w tym z innymi komponentami). Przy czym nie tworzy się obiektów samej klasy TForm (jest to tak zwana klasa abstrakcyjna), tylko wpierw tworzy się od niej konkretne klasy pochodne (np. TForm1 = TForm zawierająca TButton[1]), a następnie korzysta się z konkretnych obiektów tych klas (w tym przykładzie obiektem klasy TForm1 jest Form1).

W Delphi przyjmuje się, że każda forma (reprezentująca okienko) ma swój odrębny unit, oraz, co za tym idzie, plik .pas w którym zawarty jest jej interfejs (ang. interface) i implementacja. Dane dotyczące szczegółowych właściwości każdego z komponentów należących do formy zgromadzone są w pliku .dfm (= Delphi File Module). Prócz tego każdy program napisany w Delphi ma swój plik projektu (o rozszerzeniu .dpr = Delphi PRoject), w którym jest zawarta informacja zbiorcza o unitach wchodzących w skład programu, oraz "sekwencja rozruchowo-uruchomieniowa" dla programu. Dodatkowe dane dla programu, takie jak np. ikony lub kursory znajdują się w pliku .res (= RESource). Zatem, przykładowo, dla programu zawierającego 3 formy będziemy mieli (przynajmniej) następujące pliki:

```
program.dpr, program.res
unit1.pas, unit1.dfm
unit2.pas, unit2.dfm
unit3.pas, unit3.dfm
```

Zaś struktura logiczna programu będzie (zgrubsza) następująca:

[1] a ściślej: zawierająca TButton1, bowiem TButton (jak zresztą wiele wśród widocznych komponentów VCL) to również klasa abstrakcyjna

(klasa abstrakcyjna)TForm
–>(klasa)TForm1
–>(obiekt)Form1 zdefiniowany w pliku unit1.pas
i dookreślony w pliku unit1.dfm
–>(klasa)TForm2
–>(obiekt)Form2 zdefiniowany w pliku unit2.pas
i dookreślony w pliku unit2.dfm
–>(klasa)TForm3
–>(obiekt)Form3 zdefiniowany w pliku unit3.pas
i dookreślony w pliku unit3.dfm

2 Praktyczna realizacja: program SCRSAV

Poniżej przedstawię szczegółowe omówienie, linijka po linijce, zawartości wszystkich plików przykładowego prostego programu pod Windows, jakim jest wygaszacz ekranu. W ten sposób, poprzez naukę na konkretnym przykładzie, można w sposób bodajże najprostszy zrozumieć znaczenie podstawowych i kluczowych elementów w programowaniu w Delphi. Ponieważ znacznie powszechniejsza jest metoda uczenia poprzez wizualne projektowanie form, a dopiero później przypisywanie obsługi zdarzeń poszczególnym obiektom, skupię się tutaj na analizie *kodu źródłowego* napisanego w Delphi, nie zaś na instruktażu komponowania interfejsu (zwłaszcza iż to ostatnie jest banalne).

2.1 Specyfikacja programu wygaszacza ekranu w systemie Windows

Wygaszacz ekranu jest aplikacją systemu Windows o rozszerzeniu .scr, posiadającą swój plik konfiguracji o rozszerzeniu .ini. Pliki .scr i .ini umieszcza się w głównym katalogu Windows. Dostęp do takich opcji wygaszacza jak ustawienie hasła, lub ustawienie konfiguracji realizuje się poprzez uruchomienie programu z odpowiednimi parametrami (Windows robi to automatycznie): ’/a’, ’/c’, lub ’/s’.

2.2 Plik scrsav.dpr

Jest to główny plik programu. Pierwsza linijka programu zawiera słowo kluczowe ’program’ oraz nazwę programu. W tym przypadku jest to nazwa ’scrsav’.

```
program scrsav;
```

Poniżej w pliku znajduje się spis wszystkich unitów wykorzystywanych przez program wraz z określeniem plików położenia dla niestandardowych unitów. W tym miejscu wykorzystuje się cztery unity: Forms (standardowy), oraz Unit1, Unit2 i Unit3. W nawiasach klamrowych jako komentarz została podana przyczyna włączenia tych unitów.

```
uses
  Forms,
     Unit1 in ’Unit1.pas’ {Form1},
     Unit2 in ’Unit2.pas’ {Form2},
     Unit3 in ’Unit3.pas’ {Form3};
```

Poniższa linijka zawiera dyrektywę dla kompilatora określającą rozszerzenie (ang. Extension) nazwy pliku wykonywalnego otrzymanego po kompilacji. W tym przypadku jest to '.scr'. Zatem w wyniku procesu kompilacji otrzymamy plik 'scrsav.scr'.

```
{$E scr}
```

Następna linijka jest dyrektywą dla kompilatora określającą nazwy plików z zasobami (ang. resources), które mają być włączone do aplikacji. W plikach z zasobami (ang. resource files) znajdują się dodatkowe dane wykorzystywane bądź to przez program bądź to przez system operacyjny. Są to między innymi, w zależności od potrzeb: bitmapy, ikony, kursory, informacje o wersji, etc. W naszym przypadku plikiem z zasobami będzie plik 'scrsav.res', w którym znajduje się ikona programu (o domyślnej nazwie MAINICON).

```
{$R *.res}
```

Po informacjach dla kompilatora znajduje się już sedno pliku 'scrsav.dpr': posłużenie się obiektem Application w celu uruchomienia aplikacji. Obiekt Application jest predefiniowanym obiektem klasy TApplication, zawartej w unicie 'Forms' (właśnie z tego powodu 'Forms' znalazło się w spisie 'uses'). Przy użyciu obiektu Application dokonuje się w tym przypadku następujących czynności:

- inicjalizacji wszystkich ewentualnych podsystemów typu OLE, itp. przy pomocy metody Initialize,
- utworzenia wszystkich form programu przy pomocy metody CreateForm(TFormX, FormX), która to metoda tworzy formę typu zadanego przez TFormX (np. TForm1) i przypisuje go do referencji zadanej przez FormX (np. Form1),
- uruchomienia aplikacji przy pomocy metody Run.

W tym przypadku wygląda to następująco:

```
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.Run;
end.
```

Po wykonaniu tych linijek kodu aplikacja 'scrsav' jest uruchomiona, czyli na równi z innymi programami działającymi w danej chwili w systemie, reaguje na i (ewentualnie) generuje zdarzenia.

2.3 Unit1.pas

Poniższa linijka mówi, że mamy do czynienia z Unitem1.

```
unit Unit1;
```

Od tego miejsca zaczyna się część 'interface' unitu. Są w niej zdeklarowane stałe, zmienne, typy, funkcje i procedury, które są powszechnie (publicznie) dostępne z innych unitów.

```
interface
```

Elementy zdeklarowane w interfejsie tego unitu używają następujących unitów (jak widać, wszystkie z nich są standardowe):

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;
```

Teraz zostają zdefiniowane nowe typy: 'ustaw', oraz 'TForm1'. Typ 'ustaw' zawiera w sobie trzy pola: 'haslo', 'plik_rys', oraz 'szybkosc'. Dwa pierwsze z nich są stringami, czyli, inaczej mówiąc, napisami w których zawarte jest odpowiednio hasło wygaszacza ekranu, oraz ścieżka dostępu do rysunku, który ma się pojawiać na wygaszaczu, zaś 'szybkosc' jest polem typu byte, czyli liczbą z zakresu 0-255, determinującą szybkość przesuwania się wyświetlanego obrazka po ekranie.

```
type
  ustaw = record
     haslo,
     plik_rys : string;
     szybkosc : byte;
  end;
```

Drugim ze zdefiniowanych tutaj typów jest TForm1, czyli, w istocie, obiekt zajmujący się obsługą głównego okienka programu. Zawiera on w sobie jeden obrazek (czyli, mówiąc ściśle, obiekt klasy TImage): Image1 oraz jeden obiekt klasy TTimer. Komponent TTimer służy do obsługi zegara systemowego. Oprócz tych obiektów typ TForm1 ma zadeklarowane procedury reakcji na aktywację formy (FormActivate), reakcji na upływ czasu mierzonego zegarem systemowym (Timer1Timer), reakcji na naciśnięcie klawisza (FormKeyPress), reakcji na ruch myszką w obrębie formy (FormMouseMove), oraz reakcji na zmianę rozmiarów formy.

```
TForm1 = class(TForm)
  Image1: TImage;
  Timer1: TTimer;
  procedure FormActivate(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);

  procedure FormKeyPress(Sender: TObject; var Key: Char);
  procedure FormMouseMove(Sender: TObject; Shift: TShiftState;
     X, Y: Integer);

  procedure FormResize(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

Następnie zdefiniowane zostają używane w tym unicie zmienne globalne. 'zabezp' i 'aktywny' to zmienne typu boolowskiego, czyli mogą zawierać tylko wartości prawda-fałsz. 'zabezp' określać będzie, czy ekran jest zabezpieczony, zaś 'aktywny' określać będzie, czy wygaszacz ekranu jest aktywny. 'xx' i 'yy' to liczby shortint, czyli z zakresu -128...+127; używane one będą do przesuwania grafiki po ekranie. 'ust' to obiekt typu ustaw, zdefiniowanego powyżej w sekcji type. Zawierać on będzie w sobie ustawienia wygaszacza ekranu, zapisywane i odczytywane z pliku scrsav.ini, zaś określane przy pomocy Form3 (patrz: Unit3). 'parametr' to string charakteryzujący

zachowanie wygaszacza, zaś ini to string w którym przechowywana będzie ścieżka dostępu do pliku scrsav.ini. ’f’ to obiekt typu textfile, odpowiedzialny za odczytywanie i zapisywanie danych do pliku. ’Form1’ to oczywiście obiekt typu TForm1 (zdefiniowanego powyżej), zaś ’mx’ i ’my’ to liczby całkowite określające współrzędne kursora myszy.

```
var
  zabezp,aktywny: boolean;
  xx,yy: shortint;
  ust: ustaw;
  parametr,ini: string;
  f: textfile;
  Form1: TForm1;
  mx,my: integer;
```

Na koniec sekcji interface’u mamy deklarację funkcji ’czyt_ust’ zajmującej się odczytywaniem ustawień wygaszacza z pliku, oraz deklaracji procedury ’zapisz_ust’, zapisującej te ustawienia na dysk.

```
function czyt_ust : boolean;
procedure zapisz_ust;
```

Poniższa linijka zaczyna sekcję ’implementation’ unitu. Zawarte są tu realizacje (definicje) funkcji i procedur zdeklarowanych w sekcji ’interface’.

```
implementation
```

Implementacja korzysta z zawartości unitów ’unit2’ i ’unit3’.

```
uses unit2, unit3;
```

Ta linijka jest informacją dla kompilatora, ażeby włączył do aplikacji pliki *.dfm. W plikach tych znajdują się szczegółowe informacje na temat rozmieszczenia i ustawień komponentów na formie. Plikom tym jest poświęcony jeden z dalszych podpunktów tej pracy.

```
{$R *.dfm}
```

Funkcja ’czyt_ust’. Odczytuje ona z pliku o nazwie zawartej w zmiennej ’ini’ ustawienia wygaszacza, wpisuje je do zmiennej ’ust’. Jeżeli ta operacja udaje się, to zwraca ona wartość logiczną "prawda" (true), w przeciwnym wypadku zwraca "fałsz" (false).

```
function czyt_ust : boolean;
var
  l: string;
begin
  with ust do begin
```

Określenie domyślnych wartości dla składowych zmiennej ust:

```
haslo:=”;
plik_rys:=”;
szybkosc:=1;
```

Jeżeli plik o ścieżce dostępu takiej jak w ’ini’ istnieje...

```
if fileexists(ini) then begin
```

...to korzystając z obiektu f (wpierw go wyczyściwszy przy pomocy ’reset(f)’) wczytaj odpowiednie teksty do stringów ’haslo’ i ’plik_rys’, oraz odpowiednią liczbę do zmiennej ’szybkosc’. Procedura odczytywania opiera się na strukturze pliku scrsav.ini (opisanej w punkcie 3.6): wczytywane są kolejne napisy zawarte pomiędzy znakiem ’=’ a znakiem końca wiersza.

```
reset(f);
while (pos(’pass’,lowercase(l))=0) and not eof(f) do
   readln(f,l);
if not eof(f) then
   haslo:=copy(l,pos(’=’,l)+1,length(l));
while (pos(’file’,lowercase(l))=0) and not eof(f) do
   readln(f,l);
if not eof(f) then
   plik_rys:=copy(l,pos(’=’,l)+1,length(l));
while (pos(’speed’,lowercase(l))=0) and not eof(f) do
   readln(f,l);
if copy(l,pos(’=’,l)+1,length(l))<>” then
   szybkosc:=strtoint(copy(l,pos(’=’,l)+1,length(l)));
```

Zamknięcie pliku obsługiwanego przez ’f’ i zwrócenie "prawdy":

```
closefile(f);
czyt_ust:=true;
```

Jeżeli coś poszło nie tak, to przez funkcję zwrócona zostaje wartość "fałsz":

```
  end else
  czyt_ust:=false;
  end;
end;
```

Procedura ’zapisz_ust’ zapisuje ustawienia zawarte w zmiennej ’ust’ do pliku określonego przez ’f’:

```
procedure zapisz_ust;
begin
  rewrite(f);
  with ust do begin
     writeln(f,’[saver_settings]’);
     writeln(f,’pass=’+haslo);
     writeln(f,’file=’+plik_rys);
     writeln(f,’speed=’+inttostr(szybkosc));
```

```
  end;
  closefile(f);
end;
```

Poniższa procedura ’efekt’ zajmuje się efektem przesuwającego się po ekranie obrazka. Obrazek na ekranie reprezentowany jest przez obiekt Image1 będącym obiektem składowym Form1. Przesuwanie obrazka jest realizowane po prostu poprzez zmianę położenia Image1. Do położenie jego górnego lewego rogu jest zwiększane aż do momentu, w którym obrazek zaczyna wystawać poza ekran. Wówczas odpowiedni z mnożników (xx lub yy) zmienia znak i położenie górnego lewego rogu zaczyna się zmniejszać, aż do osiągnięcia znów którejś z krawędzi...

```
procedure efekt;
begin
  with form1.image1 do begin
     show;
     left:=left+ust.szybkosc*xx;
     top:=top+ust.szybkosc*yy;
     if (left+width>=form1.width) or (left<=0) then
     xx:=-xx;
     if (top+height>=form1.height) or (top<=0) then
     yy:=-yy;
  end;
end;
```

Ta procedura - ’FormActivate’ - jest procedurą reakcji obiektu klasy TForm1 (czyli, w naszym przypadku, Form1) na zdarzenie systemowe: aktywizację (np. kiedy użytkownik kliknie na okienko programu, czy też bezpośrednio po uruchomieniu programu).

```
procedure TForm1.FormActivate(Sender: TObject);
begin
```

Przypisanie współrzędnych kursora myszy do zmiennych ’mx’ i ’my’:

```
  mx:=mouse.CursorPos.x;
  my:=mouse.CursorPos.y;
```

Jeżeli ta aplikacja jest już raz w systemie uruchomiona i jest obecna, to zakończ działanie tej teraz uruchomionej (czyli drugiej) instancji:

```
  if hPrevInst>0 then
     application.terminate;
```

Ustalenie początkowych wartości zmiennych: mnożniki ’xx’ i ’yy’ = 1, czyli ruch rozpocznie się w prawo i do dołu, wygaszacz ma być najpierw nieaktywny, rodzaj brusha ma być przezroczysty, obrazek ma się znajdować pośrodku ekranu, plik z ustawieniami konfiguracji ma mieć taką samą ścieżkę dostępu do katalogu jak uruchomiona właśnie aplikacja, ale nazwę ’scrsav.ini’:

```
  xx:=1;
  yy:=1;
```

```
aktywny:=false;
brush.style:=bsclear;
image1.top:=(height-image1.height) div 2;
image1.left:=(width-image1.width) div 2;
ini:=extractfilepath(application.exename)+'scrsav.ini';
```

Przypisanie zmiennej 'f' obsługującej plik ustawień wygaszacza nazwy zawartej w zmiennej 'ini':

```
assignfile(f,ini);
```

Jeżeli z pliku opisanego przez 'f' nie udało się odczytać konfiguracji, to zapisz na dysk tę konfigurację, jak teraz jest ustawiona.

```
if not czyt_ust then
   zapisz_ust;
```

Jeżeli jest określony jakaś konkretna ścieżka dostępu do pliku z grafiką (do obrazka), to załaduj go z tego pliku do obiektu Image1

```
if ust.plik_rys<>" then
   image1.picture.loadfromfile(ust.plik_rys);
```

Jeżeli liczba przekazanych do programu parametrów jest większa od zera to skopiuj je do zmiennej 'parametr', a jeśli nie to niech zmienna parametr będzie pusta

```
if paramcount>0 then begin
   parametr:=copy(lowercase(paramstr(1)),1,2);
   zabezp:=false;
end else
   parametr:=";
```

Jeżeli parametrem przekazanym do programu było '/a' to wyświetl Form2 - okienko podania hasła

```
if parametr='/a' then
   form2.showmodal
else
```

Jeżeli parametrem było '/c' to wyświetl Form3 - okienko ustawień wygaszacza

```
   if parametr='/c' then
   form3.showmodal

   else begin
```

Zaś jeśli było to '/s' to ustaw zmienne 'zabezpieczenie' i 'aktywny' na prawdę.

```
   if parametr='/s' then
```

```
      zabezp:=true;
      aktywny:=true;
   end;
  end;
```

Procedura Timer1Timer. Jest to reakcja na zdarzenie "tyknięcie zegara systemowego", wywoływana jest ona co tyle milisekund, ile jest zapisane w polu "Interval" obiektu Timer1 (ustawienie to zapisane jest w pliku Unit1.dfm). Procedura ta polega na wywołaniu procedury odpowiadającej za efekt przesunięcia obrazka na ekranie, pod warunkiem, że zmienna ’aktywny’ jest ustawiona.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if aktywny then
     efekt;
end;
```

Procedura FormKeyPress - reakcji na zdarzenie naciśnięcie klawisza na klawiaturze. Jeżeli ustawiona jest zmienna ’zabezp’ to wywołane zostaje okienko Form2 - podanie hasła. Po zamknięciu okienka Form2, lub jeżeli zmienna ’zabezp’ była równa false, następuje zakończenie pracy aplikacji.

```
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if aktywny then begin
     if zabezp then
     form2.showmodal;
  application.terminate;
  end;
end;
```

Procedura FormMouseMove, czyli reakcji na ruch myszką. Jeżeli pozycja myszki okazała się być inna niż ostatnio zapisana, to następuje takie samo zachowanie (przy tych samych warunkach) co dla reakcji na wciśnięcie klawisza na klawiaturze.

```
 procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);

 begin
  if (mx<>x) or (my<>y) then
     if aktywny then begin
     if zabezp then
     form2.showmodal;
     application.terminate;
  end;
  mx:=x;
  my:=y;
 end;
```

Procedura FormResize, czyli reakcji na zmianę rozmiaru okienka. Każda próba zmiany rozmiarów okienka kończy się jego maksymalizacją.

```
procedure TForm1.FormResize(Sender: TObject);
```

```
begin
  windowstate:=wsmaximized;
end;
```

Koniec zawartości ’unit1’:

```
end.
```

2.4 Unit2.pas

Ponieważ znaczenie słów takich jak ’unit’, ’interface’, ’uses’, itp. opisałem już w poprzednim punkcie, teraz zajmę się wyłącznie rzeczami specyficznymi dla tego unitu.

```
unit Unit2;

interface
```

Tutaj warto zwrócić uwagę, że unit2 używa unitu1.

```
uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, unit1;

type
```

Deklarujemy klasę TForm2. Będzie ona odpowiadała za obsługę okienka do podawania hasła blokującego zamknięcie wygaszacza ekranu. Okienko to składa się z dwóch pól do wprowadzania tekstu: ’Edit1’ i ’Edit2’, dwóch napisów: ’Label1’ i ’Label2’, dwóch przycisków: ’Button1’ i ’Button2’, oraz jednej ozdobnej ramki ’Bevel1’. Obiekty klasy TForm2 (czyli w naszym przypadku Form2) obsługują reakcje na zdarzenia: aktywację (procedure FormActivate), kliknięcie na Button1 (procedure Button1Click), oraz na zamknięcie (procedure FormClose).

```
TForm2 = class(TForm)
  Edit1: TEdit;
  Edit2: TEdit;
  Label1: TLabel;
  Label2: TLabel;
  Button1: TButton;
  Button2: TButton;
  Bevel1: TBevel;
  procedure FormActivate(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

var
```

Tworzymy obiekt klasy ’TForm2’ nazywający się, tradycyjnie i domyślnie, ’Form2’.

```
Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.FormActivate(Sender: TObject);

begin
```

Jeżeli zbiór parametrów jest niepusty i drugą literą parametru jest ’a’ to w pierwszym edytorku (Edit1) wpisz hasło takie jakie jest zapisane w zmiennej ’ust’ (patrz punkt: ’unit1.pas’), następnie ustaw focus w tym edytorku (czyli zrób tak, żeby aktywnym elementem okienka był ten edytorek), ustaw treść napisu Label1 na ’Nowe hasło:’ i uczyń dostępnym drugi edytorek. W ten sposób - po podaniu w linii komend parametru ’/a’ ustawia się nowe hasło.

```
if (parametr<>”) and (parametr[2]=’a’) then begin
  edit1.text:=ust.haslo;
  edit1.setfocus;
  label1.caption:=’Nowe hasło:’;
  edit2.color:=clwindow;
  label2.enabled:=true;
  edit2.enabled:=true;
end else
```

W przeciwnym wypadku, jeżeli ustawiona jest zmienna ’zabezp’, wyświetlone okienko służy do sprawdzenia znajomości hasła przez użytkownika. Zatem tylko pierwsze pole ma być aktywne.

```
if zabezp then begin
  edit1.text:=”;
  edit1.setfocus;
  label1.caption:=’Hasło:’;
  edit2.color:=clbtnface;
  label2.enabled:=false;
  edit2.enabled:=false;
end else
```

W przeciwnym wypadku zakończ działanie programu.

```
  application.terminate;
end;
```

Poniżej znajduje się procedura reakcji na naciśnięcie przycisku ’Button1’. Jeżeli edytorek ’Edit2’ jest aktywny (czyli okienko służy do podawania nowego hasła), to jeżeli w obydwu edytorkach zostało wpisane to samo hasło, to nowe hasło zostaje zapisane. W innym przypadku zostaje wyświetlony komunikat o błędzie i kursor zostaje ustawiony w drugim edytorku, aby użytkownik mógł poprawić drugi wpis hasła.

```
procedure TForm2.Button1Click(Sender: TObject);
```

```
begin
  if edit2.enabled then
  if edit1.text=edit2.text then begin
  ust.haslo:=edit1.text;
     zapisz_ust;
  end else begin
  showmessage('Hasła różnią się od siebie!');
  edit2.setfocus;
  exit;
  end
else
```

Jeżeli okienko służy do sprawdzenia poprawności podanego hasła, to w przypadku podania przez użytkownika nieprawidłowego hasła następuje nic (a dokładniej: wyjście z procedury), zaś w przypadku podania prawidłowego hasła następuje zakończenie pracy programu.

```
  if edit1.text<>ust.haslo then begin
     showmessage('Niepoprawne hasło!');
     edit1.setfocus;
     exit;
  end;
  application.terminate;
end;
```

Procedura reakcji na próbę zamknięcia okienka Form2. Action=caNone powoduje akcję = niczemu, czyli okienko się nie zamyka. Dzięki temu nie można zamknąć wygaszacza bez poprawnego podania hasła.

```
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  action:=canone;
end;

end.
```

2.5 Unit3.pas

Tak jak w poprzednim punkcie, ograniczę się do rzeczy specyficznych dla tego unitu.

```
unit Unit3;

interface

uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ComCtrls, StdCtrls, ExtCtrls, ExtDlgs;

type
```

Forma opisująca to okienko (a jest to okienko służące do ustawiania ustawień wygaszacza) składa się z:

```
TForm3 = class(TForm)
```

obiektu okienka dialogowego otwarcia pliku:

```
OpenPictureDialog1: TOpenPictureDialog;
```

ozdobnych ramek:

```
Bevel1: TBevel;
Bevel2: TBevel;
Bevel3: TBevel;
```

rysunku:

```
Image1: TImage;
```

edytorków:

```
Edit1: TEdit;
Edit2: TEdit;
```

napisów:

```
Label1: TLabel;
Label2: TLabel;
```

przycisków:

```
Button1: TButton;
Button2: TButton;
Button3: TButton;
```

paska ustawień:

```
TrackBar1: TTrackBar;
```

oraz procedur reakcji na zdarzenia: (opisanych szerzej poniżej)

```
  procedure FormActivate(Sender: TObject);
  procedure TrackBar1Change(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  procedure Button3Click(Sender: TObject);
  procedure Button1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

  var
    Form3: TForm3;

  implementation

  uses unit1;

  {$R *.dfm}
```

Procedura reakcji na aktywację formy:

```
  procedure TForm3.FormActivate(Sender: TObject);
  begin
    with ust do begin
```

Jeżeli w ustawieniach jest już zapisany jakiś rysunek to załaduj go i wyświetl:

```
    if plik_rys<>” then begin
       edit1.text:=plik_rys;
       image1.picture.loadfromfile(plik_rys);
    end else
```

Jeśli nie ma żadnego rysunku to w edytorku wpisz tekst informujący o tym:

```
       edit1.text:=’[brak]’;
```

Ustaw odpowiednią pozycję na pasku odpowiadającą zapisanej w ustawieniach szybkości przesuwania się obrazka po ekranie:

```
       trackbar1.position:=szybkosc;
       edit2.text:=inttostr(szybkosc);
    end;
  end;
```

Procedura reakcji na zmianę położenia wskaźnika na pasku ustawień. Reakcja polega na wypisanie w edytorku Edit2 liczby odpowiadającej ustawieniu wskaźnika.

```
  procedure TForm3.TrackBar1Change(Sender: TObject);
  begin
    edit2.text:=inttostr(trackbar1.position);
  end;
```

Procedura reakcji na wciśnięcie przycisku Button2: zapisanie ustawień i zakończenie pracy aplikacji.

```
procedure TForm3.Button2Click(Sender: TObject);
begin
  with ust do begin
     plik_rys:=edit1.text;
     szybkosc:=trackbar1.position;
  end;
  zapisz_ust;
  application.terminate;
end;
```

Procedura reakcji na wciśnięcie przycisku Button3: zakończenie pracy aplikacji bez zapisania dokonanych zmian.

```
procedure TForm3.Button3Click(Sender: TObject);
begin
  application.terminate;
end;
```

Procedura reakcji na wciśnięcie przycisku Button1: wyświetlenie okienka dialogowego otwarcia pliku graficznego i ewentualne załadowanie obrazka sprecyzowanego przez użytkownika w tym okienku (jeżeli taki wybór został przez niego w ogóle dokonany).

```
procedure TForm3.Button1Click(Sender: TObject);
begin
  with openpicturedialog1 do
     if execute then begin
     edit1.text:=filename;
     image1.picture.loadfromfile(filename);
  end;
end;
```

Procedura reakcji na wciśnięcie przycisku zamknięcia okienka. Zamknięcie okienka powoduje zakończenie pracy aplikacji.

```
procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  application.terminate;
end;

end.
```

2.6 Pliki *.dfm

W plikach *.dfm zawarta jest informacja na temat właściwości wszystkich komponentów umieszczonych na formie, o ile te właściwości różnią się od domyślnych. Dla przykładu zamieściłem tu zawartość pliku unit1.dfm. Pozostałe wyglądają podobnie. Zawartość plików .dfm ustala się automatycznie w fazie wizualnego projektowania aplikacji i zazwyczaj nie ma potrzeby wgłębiać się bezpośrednio w ich zawartość.

```
object Form1: TForm1
  Left = 242
```

```
  Top = 155
  Align = alClient
  BorderIcons = []
  BorderStyle = bsNone
  Caption = 'Wygaszacz ekranu'
  ClientHeight = 280
  ClientWidth = 484
  Color = clBtnFace
  TransparentColor = True
  TransparentColorValue = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  WindowState = wsMaximized
  OnActivate = FormActivate
  OnKeyPress = FormKeyPress
  OnMouseMove = FormMouseMove
  OnResize = FormResize
  PixelsPerInch = 96
  TextHeight = 13
  object Image1: TImage
     Left = 8
     Top = 48
     Width = 177
     Height = 177
     Stretch = True
     Visible = False
  end
  object Timer1: TTimer
     Interval = 50
     OnTimer = Timer1Timer
     Left = 8
     Top = 8
  end
end
```

2.7 Plik scrsav.ini

```
[saver_settings]
pass=Brey038f2
file=c:\windows\psizab.bmp
speed=1
```

2.8 Plik scrsav.res

Plik ten zawiera ikonkę programu.