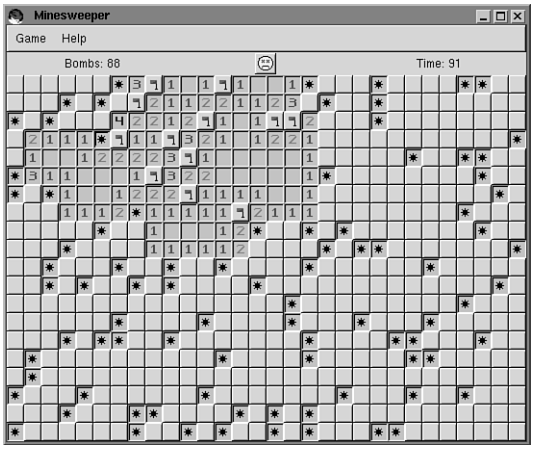
# Rozdział 9

## Saper

Saper jest prostą grą, która zyskała popularność, kiedy Microsoft dołączył ją do systemu Windows. Celem gry jest odsłonięcie wszystkich znajdujących się na planszy pól, pod którymi nie ma bomb. Pierwsze próby są dość przypadkowe, ale odsłonięte kwadraty dostarczają pewnych wskazówek, które pomagają w rozbrojeniu reszty pól. Każde pole „wie", ile bomb znajduje się w otaczających je polach, więc odkrycie pola bez bomby pozwala wywnioskować, którzy z jego sąsiadów mogą zawierać śmiercionośny ładunek. Aby zabawa była bardziej interesująca, zegar pokazuje nam, ile czasu zajęła gra, więc po wyczyszczeniu planszy możemy spróbować ukończyć grę w krótszym czasie. Planszę Sapera można obejrzeć na rysunku 9.1.




**Rysunek 9.1**. Saper.

Program Sapera uwypukla zagadnienia, którymi zajmowaliśmy się do tej pory. Prawdę powiedziawszy, szkielet programu bardzo przypomina aplikację kalkulatora. Oba programy wykorzystują przyciski w tabeli pakującej, ale Saper jest nieco bardziej skomplikowany. Posiada menu,

zawierające opcje ustawiające poziom trudności (początkujący, średni, zaawansowany) oraz pasek narzędziowy, który wyświetla zegar i liczbę nie odkrytych bomb. Saper wykorzystuje także rysunki xpm przedstawiające bomby, uśmiechnięty przycisk, kolorowe cyfry i flagi.

Wszystkie przyciski przechowywane są w dwuwymiarowej tablicy elementów typu typPrzyciskSapera. Struktura ta przechowuje wskaźnik do kontrolki przycisku oraz informacje o tym, czy pod przyciskiem jest bomba i ile sąsiadów przycisku ukrywa bomby.

```
/*
 * --- struktura danych, przechowująca przyciski Sapera
 */
typedef struct {
   int         stanPrzycisku;        /* bieżący stan przycisku */
   GtkWidget  *kontrolka;            /* uchwyt do przycisku */
   int         nBombyWPoblizu;       /* ile bomb jest wokół pola? */
   int         bMaBombe;             /* czy pod przyciskiem jest bomba? */
   int         nWiersz;              /* rząd tabeli pakującej */
   int         nKol;                 /* kolumna tabeli pakującej */
} typPrzyciskSapera;
```

Kiedy użytkownik rozpocznie grę, komputer szybko generuje planszę, rozmieszczając właściwą liczbę bomb pod niektórymi przyciskami. Oblicza także, ile bomb przylega do każdego pola planszy. Informacja ta jest początkowo ukrywana przed użytkownikiem, ale po kliknięciu na przycisku, pod którym nie ma bomby, jest ona wyświetlana na przycisku.

Najpierw trzeba stworzyć dane dla rysunków. Rysunki są proste i intuicyjne. Składają się z kilku uśmiechniętych twarzy, które zostaną umieszczone na przycisku paska narzędziowego, flagi, przy pomocy której użytkownik może zaznaczać położenie bomb, oraz kolorowych cyfr, które wyświetlają liczbę bomb wokół danego pola.

## bitmapy.h

Rysunki wykorzystywane przez Sapera (oprócz kolorowych cyfr) znajdują się w pliku bitmaps.h. Zwróćmy uwagę, że twarz ma trzy różne rysunki - jeden wyświetlany w czasie gry, jeden w razie przegranej, a trzeci w przypadku zwycięstwa. Nie wymagają zbyt wiele kodu, a dodają grze trochę charakteru.

```
/*
 * --- Flaga do zaznaczania bomb
```

```
 */
static char *xpm_flaga[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"            ",
"  RRRRRRR   ",
"  RRRRR r r   ",
"  RRR r r r r   ",
"  R r r r r r r   ",
"        X   ",
"        X   ",
"        X   ",
"        X   ",
"        X   ",
"       XXX  ",
"            ",
};

/*
 * --- Bomba. Cóż, każdemu trafia się gorszy dzień.
 */
static char *xpm_bomba[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"            ",
"     X      ",
"  X  X  X   ",
"    XXXXX    ",
"    XXXXX    ",
" XXXXXXXXX  ",
"    XXXXX    ",
"    XXXXX    ",
"  X  X  X   ",
"     X      ",
"            ",
```

```
"            ",
};

/*
 * --- Zły ruch!
 */
static char *xpm_duzex[] = {
"12 12 4 1",
"  c None",
"X c #000000",
"R c #FF0000",
"r c #AA0000",
"RRR      RRR",
" RRR    RRR ",
"  RRR  RRR  ",
"   RRRRRR   ",
"    RRRR    ",
"    RRRR    ",
"    RRRR    ",
"   RRRRRR   ",
"  RRR  RRR  ",
" RRR    RRR ",
"RRR      RRR",
"            ",
};

/*
 * --- Bitmapa z uśmiechem
 */
static char *xpm_usmiech[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
"r c #AA0000",
"      ......      ",
"   ..XXXXXX..    ",
" ..XXXXXXXXXX.   ",
" .XXXXXXXXXXXX. ",
" .XX..XXXX..XX. ",
".XXX..XXXX..XXX.",
```

```
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XX.XXXXXX.XX. ",
" .XXX......XXX. ",
"  .XXXXXXXXXX.  ",
"   ..XXXXXX..   ",
"     ......     ",
"                ",
};

/*
 * --- Smutna mina. Przegrałeś.
 */
static char *xpm_smutek[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
"r c #AA0000",
"     ......     ",
"   ..XXXXXX..   ",
" ..XXXXXXXXXX.  ",
" .XXXXXXXXXXXX. ",
" .XX.X.XX.X.XX. ",
".XXXX.XXXX.XXXX.",
".XXX.X.XX.X.XXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XXX......XXX. ",
" .XX.XXXXXX.XX. ",
"  .XXXXXXXXXX.  ",
"   ..XXXXXX..   ",
"     ......     ",
"                ",
};

/*
 * --- Mamy zwyciêzcê.
```

```
 */
static char *xpm_wygrana[] = {
"16 16 4 1",
"  c None",
". c #000000",
"X c #FFFF00",
"r c #AA0000",
"     ......     ",
"   ..XXXXXX..   ",
" ..XXXXXXXXXX.  ",
" .XXXXXXXXXXXX. ",
" .XX...XX...XX. ",
".XX.........XX.",
".X.X...XX...X.X.",
"..XXXXXXXXXXXX..",
".XXXXXXXXXXXXXX.",
".XXXXXXXXXXXXXX.",
" .XX.XXXXXX.XX. ",
" .XXX......XXX. ",
"  .XXXXXXXXXX.  ",
"   ..XXXXXX..   ",
"     ......     ",
"                ",
};
```

## cyfry.h

Cyfry, oznaczające liczbę bomb wokół danego pola, są piksmapami. Ponieważ wszystkie dane dla przycisków znajdują się w piksmapach, procedury umieszczające na przyciskach bomby, flagi czy cyfry można zawrzeć w pojedynczym fragmencie kodu - zmienia się tylko rysunek.

```
/*
1 - jasnoniebieski
2 - zielony
3 - czerwony
4 - ciemnoniebieski
5 - brązowo-fioletowy
*/

static const char *xpm_jeden[] = {
```

```
"12 12 2 1",
"  c None",
"X c #3333CC",
"            ",
"     XX     ",
"    XXX     ",
"   X XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"     XX     ",
"   XXXXXX   ",
"            ",
"            ",
};

static const char *xpm_dwa[] = {
"12 12 2 1",
"  c None",
"X c #009900",
"            ",
"   XXXXXX   ",
"  X      X  ",
"        XX  ",
"       XX   ",
"      XX    ",
"     XX     ",
"    XX      ",
"   XX       ",
"  XXXXXXXX  ",
"            ",
"            ",
};

static const char *xpm_trzy[] = {
"12 12 2 1",
"  c None",
"X c #AA0000",
"            ",
"   XXXXX    ",
```

```
"       XX  ",
"       XX  ",
"   XXXXXX   ",
"       XX  ",
"       XX  ",
"       XX  ",
"       XX  ",
"  XXXXXX    ",
"             ",
"             ",
};

static const char *xpm_cztery[] = {
"12 12 2 1",
"  c None",
"X c #000066",
"             ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"  XXXXXXXX  ",
"       XX  ",
"       XX  ",
"       XX  ",
"       XX  ",
"             ",
"             ",
};

static const char *xpm_piec[] = {
"12 12 2 1",
"  c None",
"X c #992299",
"             ",
"  XXXXXXXX  ",
"  XX        ",
"  XX        ",
"  XXXXXXX   ",
"       XX  ",
"       XX  ",
```

```
"       XX  ",
"  XX    XX  ",
"  XXXXXXX   ",
"            ",
"            ",
};

static const char *xpm_szesc[] = {
"12 12 2 1",
"  c None",
"X c #550055",
"            ",
"   XXXXXX   ",
"  XX        ",
"  XX        ",
"  XXXXXXX   ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"   XXXXXX   ",
"            ",
"            ",
};

static const char *xpm_siedem[] = {
"12 12 2 1",
"  c None",
"X c #550000",
"            ",
"  XXXXXXXX  ",
"       XX  ",
"      XX   ",
"      XX   ",
"     XX    ",
"     XX    ",
"    XX     ",
"    XX     ",
"    XX     ",
"            ",
"            ",
```

```
};

static const char *xpm_osiem[] = {
"12 12 2 1",
"  c None",
"X c #441144",
"            ",
"   XXXXXX   ",
"  XX    XX  ",
"  XX    XX  ",
"   XXXXXX   ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"  XX    XX  ",
"   XXXXXX   ",
"            ",
"            ",
};
```

## zegar.c

Gra musi przechowywać i uaktualniać liczbę sekund, które upłynęły od kliknięcia przez użytkownika pierwszego przycisku na planszy. Funkcja StartZegara jest wywoływana po pierwszym kliknięciu przycisku - prawdę powiedziawszy, jest wywoływana po *każdym* kliknięciu przycisku, ale jeśli zegar jest już uruchomiony, funkcja ignoruje wywołanie. Funkcja StartZegara uruchamia funkcję zwrotną (FunkcjaZwrotnaZegara), która będzie wywoływana co sekundę w celu uaktualnienia i wyświetlenia bieżącego czasu. Natomiast funkcja KoniecZegara wywoływana jest wtedy, kiedy gracz trafi na bombę, wygra grę, albo rozpocznie ją od nowa, klikając na uśmiechniętym przycisku.

```
/*
 * zegar.c
 *
 * Autor: Eric Harlow
 *
 * Procedury do uaktualniania liczby sekund
 */

#include <gtk/gtk.h>
```

```
static int nSekundy = 0;
static gint zegar = 0;
static int bZegarPracuje = FALSE;

void UaktualnijSekundy (int);

/*
 * FunkcjaZwrotnaZegara
 *
 * Będzie wywoływana co sekundę, aby uaktualnić liczbę
 * sekund w polu zegara.
 */
gint FunkcjaZwrotnaZegara (gpointer dane)
{
   /* --- Upłynęła kolejna sekunda --- */
   nSekundy++;

   UaktualnijSekundy (nSekundy);
}

/*
 * StartZegara
 *
 * Rozpoczyna odliczanie, kiedy użytkownik kliknie
 * pierwszy przycisk.
 */
void StartZegara ()
{

   /* --- Jeśli zegar jeszcze nie pracuje... --- */
   if (!bZegarPracuje) {

      /* --- ...zaczynamy od zera --- */
      nSekundy = 0;

      /* --- Wywołujemy FunkcjęZwrotnąZegara co 1000ms --- */
      zegar = gtk_timeout_add (1000, FunkcjaZwrotnaZegara, NULL);

      /* --- Zegar pracuje --- */
      bZegarPracuje = TRUE;
   }
}
```

```
/*
 * KoniecZegara
 *
 * Zatrzymuje zegar. Może gracz trafił na bombę.
 */
void KoniecZegara ()
{
   /* --- Jeśli zegar pracuje... --- */
   if (bZegarPracuje) {

      /* --- ...zatrzymujemy zegar --- */
      gtk_timeout_remove (zegar);

       /* --- Ustawiamy odpowiednio znacznik --- */
      bZegarPracuje = FALSE;
   }
}
```

## saper.c

Gra składa się z siatki N x M przycisków - przełączników (GtkToggleButton), reprezentowanych przez strukturę typPrzyciskSapera. Każdemu przyciskowi przypisujemy rząd (nWiersz) oraz kolumnę (nKol) w siatce, a także znacznik, który wskazuje, czy pod przyciskiem znajduje się bomba (bMaBombe). Oprócz tego struktura przechowuje wskaźnik do utworzonego przycisku oraz licznik bomb, znajdujących się dookoła przycisku (używany tylko wtedy, kiedy pod przełącznikiem nie ma bomby). Znacznik stanPrzycisku odzwierciedla jeden ze stanów, w których może znajdować się przycisk. Zaczynamy od stanu PRZYCISK_NIEZNANY, ale użytkownik może zmienić stan przycisku na PRZYCISK_FLAGA, klikając go prawym klawiszem myszy. Czynność ta powoduje umieszczenie na przycisku rysunku flagi. Stan PRZYCISK_WLACZONY oznacza, że przycisk został wciśnięty. Jeśli pod przyciskiem znajduje się bomba, stan ten doprowadzi do zakończenia gry. Ostatni stan (PRZYCISK_NIEPEWNY) pozwala użytkownikowi zaznaczyć pole jako wątpliwe. W takim przypadku na przycisku pojawia się znak zapytania. Ten stan przycisku nie jest chwilowo zaimplementowany, ale nietrudno go dodać; pozostawiamy to jako ćwiczenie dla czytelników.

Odrobina rekurencji upraszcza kodowanie. Jeśli na przykład użytkownik kliknie przycisk bez bomby, dookoła którego również nie ma żadnych bomb, wówczas pobliskie pola są rekurencyjnie odkrywane tak, jakby

same zostały kliknięte. Jeśli któryś z nich także nie sąsiaduje z żadnymi bombami, odkrywane są wszystkie pola dookoła niego itd.

Podczas tworzenia przycisków każdemu z nich przypisujemy tę samą funkcję zwrotną, wywoływaną w przypadku kliknięcia. Parametr danych dla funkcji zwrotnej stanowi struktura danych przycisku, pochodząca z tablicy struktur typPrzyciskSapera. Dane, które otrzymuje funkcja zwrotna, dotyczą tylko kliknięty przycisku, ale ponieważ zawierają one rząd i kolumnę przycisku, można na ich podstawie określić także jego sąsiadów.

```
/*
 * Plik: saper.c
 * Autor: Eric Harlow
 *
 * Program przypomina Sapera z Windows
 */

#include <gtk/gtk.h>
#include <stdlib.h>
#include <time.h>
#include "rozne.h"
#include "bitmapy.h"
#include "cyfry.h"

void UstawIkonePrzyciskuStart (gchar **xpm_dane);

/*
 * --- Ustalamy rozmiar przycisków
 */
#define SZEROKOSC_PRZYCISKU 18
#define WYSOKOSC_PRZYCISKU 19

/*
 * --- Każda z cyfr, pokazujących liczbę bomb jest kolorową
 *     bitmapą. Poniższa tablica znacznie przyspiesza
 *     wyszukiwanie cyfr.
 */
gpointer cyfry[] = {
   NULL,
   xpm_jeden,
   xpm_dwa,
   xpm_trzy,
   xpm_cztery,
```

```
    xpm_piec,
    xpm_szesc,
    xpm_siedem,
    xpm_osiem
};

/*
 * --- To są dostępne stany przycisków
 *
 * NIEZNANY - Przycisk pusty. Nie wiadomo, co się pod nim kryje
 * FLAGA    - Użytkownik umieścił na przycisku flagę, podejrzewając
 *            że pod nim jest bomba
 * NIEPEWNY - Stan nie zaimplementowany
 * WLACZONY - Przycisk został wciśnięty
 */
enum {
    PRZYCISK_NIEZNANY,
    PRZYCISK_FLAGA,
    PRZYCISK_NIEPEWNY,
    PRZYCISK_WLACZONY
};

/*
 * --- struktura danych, przechowująca przyciski Sapera
 */
typedef struct {
    int          stanPrzycisku;      /* bieżący stan przycisku */
    GtkWidget  *kontrolka;           /* uchwyt do przycisku */
    int          nBombyWPoblizu;     /* ile bomb jest wokół pola? */
    int          bMaBombe;           /* czy pod przyciskiem jest bomba? */
    int          nWiersz;            /* rząd tabeli pakującej */
    int          nKol;               /* kolumna tabeli pakującej */
} typPrzyciskSapera;

/*
 * --- Domyślne wartości rozmiarów siatki
 *     i liczby bomb.
 */
static int nLWierszy = 10;
static int nLKolumn = 10;
static int nLiczbaBomb = 10;
```

```
/*
 * --- Siatkę tworzy dwuwymiarowa tablica. To są
 *     maksymalne rozmiary.
 */
#define MAKS_WIERSZY 35
#define MAKS_KOLUMN 35
/*
 * Zmienne globalne
 */

/* --- Przypisujemy tablicy maksymalne rozmiary. Właściwie powinno się
 *     robić to dynamicznie, ale siatkę najłatwiej zaimplementować
 *     jako zwykłą dwuwymiarową tablicę
 */
typPrzyciskSapera plansza[MAKS_KOLUMN][MAKS_WIERSZY];

/* --- Znaczniki używane przez grę --- */
int bKoniecGry = FALSE;            /* --- Gra skończona? --- */
int bOdPoczatku = FALSE;           /* --- Gra skończona? --- */
int nPozostaleBomby;               /* --- Nie odkryte bomby --- */

GtkWidget *tabela = NULL;          /* --- Tabela z siatką przycisków --- */
GtkWidget *przycisk_start;         /* --- Przycisk zaczynający grę --- */
GtkWidget *etykieta_bomby;         /* --- Etykieta z liczbą bomb --- */
GtkWidget *etykieta_czas;          /* --- Etykieta z czasem --- */
GtkWidget *ypole;                  /* --- Pole pakujące w głównym oknie - */

/*
 * --- Prototypy
 */
void PokazUkryteInformacje (typPrzyciskSapera *pole);
void UtworzPrzyciskiSapera (GtkWidget *tabela, int k, int w, int znacznik);
void FZwrotnaZwolnijPotomka(GtkWidget *kontrolka);

/*
 * SprawdzWygrana
 *
 * Sprawdza, czy gracz wygrał. Wygrana polega na tym, że liczba nie
 * odkrytych pól jest równa liczbie bomb.
 */
void SprawdzWygrana ()
{
```

```
    int i, j;
    int nBomby = 0;

    /* --- Przechodzimy przez wszystkie pola --- */
    for (i = 0; i <  nLKolumn; i++) {
        for (j = 0; j < nLWierszy; j++) {

            /* --- Jeśli przycisk jest nie wciśnięty
                   albo opatrzony flagą... --- */
            if (plansza[i][j].stanPrzycisku == PRZYCISK_NIEZNANY ||
                plansza[i][j].stanPrzycisku == PRZYCISK_FLAGA) {

                /* --- ...to może być pod nim bomba. --- */
                nBomby ++;
            }
        }
    }

    /* --- Czy jest tyle bomb, ile nie odkrytych pól? --- */
    if (nBomby == nLiczbaBomb) {

        /* --- Koniec gry. Mamy zwycięzcę. --- */
        KoniecZegara ();
        UstawIkonePrzyciskuStart ((gchar **) xpm_wygrana);
        bKoniecGry = TRUE;
    }
}

/*
 * UmiescIkoneNaPrzycisku
 *
 * Zmienia ikonę na przycisku na jeden z rysunków xpm
 *
 * pole - kwadrat siatki
 * xpm - dane rysunku
 */
void UmiescIkoneNaPrzycisku (typPrzyciskSapera *pole, char *xpm[])
{
    GtkWidget *kontrolka;

    /* --- tworzymy kontrolkę z danych xpm --- */
    kontrolka = UtworzKontrolkeZXpm (tabela, (gchar **) xpm);
```

```
    /* --- Umieszczamy piksmapę na rysunku --- */
    gtk_container_add (GTK_CONTAINER (pole->kontrolka), kontrolka);

    /* --- usuwamy odniesienie do rysunku, aby została usunięta
           wraz z przyciskiem --- */
    gdk_pixmap_unref ((GdkPixmap *) kontrolka);
}

/*
 * UaktualnijSekundy
 *
 * Uaktualnia licznik sekund na pasku narzędziowym
 *
 * nSekundy - ile sekund należy wyświetlić
 */
void UaktualnijSekundy (int nSekundy)
{
    char bufor[44];

    /* --- Zmieniamy etykietę, aby wskazywała bieżący czas --- */
    sprintf (bufor, "Czas: %d", nSekundy);
    gtk_label_set (GTK_LABEL (etykieta_czas), bufor);
}

/*
 * PokazLiczbeBomb
 *
 * Pokazuje liczbę pozostałych bomb.
 */
void PokazLiczbeBomb ()
{
    char bufor[33];

    /* --- Zostało XX bomb --- */
    sprintf (bufor, "Bomby: %d", nPozostaleBomby);
    gtk_label_set (GTK_LABEL (etykieta_bomby), bufor);
}

/*
 * ZwolnijPotomka
 *
 * Zwalnia wszystkich potomków kontrolki
 * Wywoływana wtedy, kiedy na przycisku trzeba umieścić nowy
```

```
 * rysunek. Stary rysunek jest usuwany.
 */
void ZwolnijPotomka (GtkWidget *kontrolka)
{
   /* --- Zwalniamy potomków przycisku --- */
   gtk_container_foreach (
            GTK_CONTAINER (kontrolka),
            (GtkCallback) FZwrotnaZwolnijPotomka,
            NULL);

}

/*
 * delete_event
 *
 * Okno jest zamykane, trzeba zakończyć pętlę GTK
 *
 */
void delete_event (GtkWidget *kontrolka, gpointer *dane)
{
   gtk_main_quit ();
}

/*
 * PokazBomby
 *
 * Kliknięto na bombie, więc trzeba pokazać, gdzie rzeczywiście były
 * bomby (przynajmniej te, których dotąd nie odkryto). Wyświetlamy
 * nie znalezione bomby oraz te, które niby zostały znalezione, ale
 * wcale nie były bombami.
 */
void PokazBomby (typPrzyciskSapera *trafionobombe)
{
   int i, j;
   typPrzyciskSapera *pole;
   GtkWidget *kontrolka_x;

   /* --- Przeglądamy wszystkie pola --- */
   for (i = 0; i <  nLKolumn; i++) {
      for (j = 0; j < nLWierszy; j++) {

         /* --- Pobieramy strukturę danych --- */
```

```
            pole = &plansza[i][j];
            /* --- Jeśli jest tutaj przycisk, a pod nim --- */
            /* --- jest bomba... --- */
            if (pole->stanPrzycisku == PRZYCISK_NIEZNANY &&
               pole->bMaBombe) {

               /* --- ...wyświetlamy bombę --- */
               PokazUkryteInformacje (pole);

            /* --- Jeśli zaznaczono pole flagą, a nie ma
             *     w nim bomby... --- */
            } else if (pole->stanPrzycisku == PRZYCISK_FLAGA &&
                     !pole->bMaBombe) {

               /* --- ...usuwamy flagę... --- */
               ZwolnijPotomka (pole->kontrolka);

               /* --- ...i pokazujemy w tym miejscu X --- */
               UmiescIkoneNaPrzycisku (pole, xpm_duzex);
            }
         }
      }
   }

   /*
    * OdkryjSasiedniePola
    *
    * Odkrywa wszystkie pola wokół danego pola.
    *
    * kol, wiersz - pozycja, wokół której należy odkryć pola
    * Odkrywa wszystkie pola wokół oznaczonego niżej przez X:
    *
    *        |------|------|------
    *        |      |      |     |
    *        ---------------------
    *        |      |  X   |     |
    *        ---------------------
    *        |      |      |     |
    *        |------|------|------
    */
   void OdkryjSasiedniePola (int kol, int wiersz)
   {
```

```
    int i, j;

    /* --- Sprawdzamy kolumnę wstecz i kolumnę do przodu --- */
    for (i = MAX (kol-1, 0); i <= MIN (kol+1, nLKolumn-1); i++) {

        /* --- Sprawdzamy wiersz powyżej i poniżej --- */
        for (j = MAX (wiersz-1, 0); j <= MIN (wiersz+1, nLWierszy-1); j++) {

            /* --- Wyświetlamy to, co jest pod spodem --- */
            PokazUkryteInformacje (&plansza[i][j]);
        }
    }
}

/*
 * PokazUkryteInformacje
 *
 * Pokazuje, co znajduje się pod przyciskiem.
 * Może to być bomba, albo pole, wyświetlające liczbę bomb
 * znajdujących się dookoła.
 * Może to być puste pole.
 */
void PokazUkryteInformacje (typPrzyciskSapera *pole)
{
    char      bufor[88];
    GtkWidget *kontrolka;

    /* --- Jeśli przycisk jest już wciśnięty, wracamy --- */
    if (pole->stanPrzycisku == PRZYCISK_WLACZONY) {
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON
                                (pole->kontrolka), TRUE);
        return;
    }

    /* --- Jeśli przycisk jest opatrzony flagą, to nic nie
           ujawniamy  --- */
    if (pole->stanPrzycisku == PRZYCISK_FLAGA) {

        /* --- Gracz twierdzi, że jest tu bomba - nie odkrywamy
         *     więc pola, choćby nie mogło być w nim bomby.
         */
        gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON
                                (pole->kontrolka), FALSE);
```

```
    } else {

      /* --- Ustawiamy stan przycisku na WŁĄCZONY --- */
      pole->stanPrzycisku = PRZYCISK_WLACZONY;
      gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON
                        (pole->kontrolka), TRUE);

      /* --- Jeśli w tym polu jest bomba --- */
      if (pole->bMaBombe) {

        /* --- Pokazujemy bombę w polu --- */
        UmiescIkoneNaPrzycisku (pole, xpm_bomba);

      /* --- Nie ma bomby, ale są w pobliżu --- */
      } else if (pole->nBombyWPoblizu) {

        /* --- Pokazujemy liczbę sąsiadujących bomb --- */
        UmiescIkoneNaPrzycisku(pole, cyfry[pole->nBombyWPoblizu]);

      } else {

        /* --- Hmm. Kliknięto pole bez bomby i bez  --- */
        /*     sąsiadujących bomb. Odkrywamy wszystkie --- */
        /*     pola dookoła (być może lawinowo) --- */
        OdkryjSasiedniePola (pole->nKol, pole->nWiersz);
      }
    }
}

/*
 * NowaGra
 *
 * Zerujemy grę, aby można było zagrać od nowa. Ustawiamy licznik
 * bomb i wyświetlamy pustą planszę.
 */
void NowaGra (int nKolumnySiatki, int nWierszeSiatki,
          int nBomby, int bNowePrzyciski)
{

    /* --- Ustawiamy liczbę bomb w siatce --- */
    nLiczbaBomb = nBomby;

    /* --- Ustawiamy liczbę nie odkrytych bomb --- */
    nPozostaleBomby = nBomby;
```

```
    /* --- Tworzymy przyciski sapera. --- */
    UtworzPrzyciskiSapera (tabela, nKolumnySiatki,
                           nWierszeSiatki, bNowePrzyciski);

    /* --- Zatrzymujemy zegar --- */
    KoniecZegara ();

    UaktualnijSekundy (0);

    UstawIkonePrzyciskuStart ((gchar **) xpm_usmiech);
}

/*
 * FZwrotnaZwolnijPotomka
 *
 * Zwalnia kontrolkę.
 */
void FZwrotnaZwolnijPotomka(GtkWidget *kontrolka)
{
    gtk_widget_destroy (kontrolka);
}

/*
 * UstawIkonePrzyciskuStart
 *
 * Ustawia ikonę przycisku rozpoczynającego grę.
 * Zazwyczaj będzie to albo wesoła, albo smutna twarz.
 */
void UstawIkonePrzyciskuStart (gchar **xpm_dane)
{
    GtkWidget *kontrolka;

    /* --- Tworzymy kontrolkę z xpm --- */
    kontrolka = UtworzKontrolkeZXpm (przycisk_start, xpm_dane);

    /* --- Zwalniamy wszystkich potomków przycisku --- */
    ZwolnijPotomka (przycisk_start);

    /* --- Dodajemy rysunek do przycisku --- */
    gtk_container_add (GTK_CONTAINER (przycisk_start), kontrolka);
}

/*
 * kliknieto_start
```

```
 *
 * Procedura obsługi zdarzenia, wywoływana po kliknięciu
 * przycisku rozpoczynającego grę.
 */
void kliknieto_start (GtkWidget *kontrolka, gpointer *dane)
{
   UstawIkonePrzyciskuStart ((gchar **) xpm_usmiech);
   NowaGra (nLKolumn, nLWierszy, nLiczbaBomb, FALSE);
}

/*
 * kliknieto_plansze
 *
 * Procedura obsługi zdarzenia, wywoływana po kliknięciu
 * któregoś z przycisków na planszy.
 *
 * kontrolka - kliknięty przycisk.
 * dane - struktura przycisku.
 */
void kliknieto_plansze (GtkWidget *kontrolka, gpointer *dane)
{
   typPrzyciskSapera *pole;
   GtkWidget *etykieta;

pole = (typPrzyciskSapera *) dane;

   /* --- Jeśli gra jest skończona --- */
   if (bKoniecGry) {

      /* --- Zostawiamy przycisk tak, jak był. --- */
      gtk_toggle_button_set_state (GTK_TOGGLE_BUTTON (kontrolka),
                  (pole->stanPrzycisku == PRZYCISK_WLACZONY));
      return;
   }

/* --- Jeśli gra jest zerowana --- */
   if (bOdPoczatku) return;

   /* --- Zaczynamy odliczanie czasu --- */
   StartZegara ();

   /* --- Jeśli kliknięto bombę... --- */
   if (pole->bMaBombe) {
```

```
        /* --- Koniec gry! --- */
        bKoniecGry = TRUE;

        /* --- Twarz się zasmuca. --- */
        UstawIkonePrzyciskuStart ((gchar **) xpm_smutek);

        /* --- Wyświetlamy wszystkie nie odkryte bomby. --- */
        KoniecZegara ();
        PokazBomby (pole);

    } else {

        /* --- tworzymy etykietę dla przycisku i wyświetlamy --- */
        /* --- na niej liczbę sąsiadujących bomb. --- */
        PokazUkryteInformacje (pole);
        SprawdzWygrana ();
    }
}

/*
 * kliknieto_przycisk
 *
 * Użytkownik mógł kliknąć planszę prawym klawiszem myszy.
 * W takim przypadku należy odpowiednio zmienić rysunek
 * na przycisku.
 */
void kliknieto_przycisk (GtkWidget *kontrolka,
                         GdkEventButton *zdarzenie, gpointer *dane)
{
    typPrzyciskSapera *pole;
    GtkWidget *kontrolkaPiksmapy;

    /* --- Ignorujemy zdarzenie, jeśli gra jest już skończona --- */
    if (bKoniecGry) {
        return;
    }

    /* --- Które pole kliknięto? --- */
    pole = (typPrzyciskSapera *) dane;

    /* --- Upewniamy się, że zdarzenie to kliknięcie przycisku --- */
    if (zdarzenie->type == GDK_BUTTON_PRESS) {

        /* --- Czy był to prawy klawisz myszy? --- */
```

```
        if (zdarzenie->przycisk == 3) {

            switch (pole->stanPrzycisku) {

                case PRZYCISK_NIEZNANY:

                    /* --- Zwalniamy potomków przycisku --- */
                    ZwolnijPotomka (kontrolka);

                    pole->stanPrzycisku = PRZYCISK_FLAGA;
                    UmiescIkoneNaPrzycisku (pole, xpm_flaga);
                    nPozostaleBomby --;
                    break;
                case PRZYCISK_FLAGA:

                    /* --- Zwalniamy potomków przycisku --- */
                    ZwolnijPotomka (kontrolka);

                    pole->stanPrzycisku = PRZYCISK_NIEZNANY;
                    nPozostaleBomby ++;
                    break;
            }
            PokazLiczbeBomb ();
            SprawdzWygrana ();
        }
    }
}

/*
 * UtworzPrzycisk
 *
 * Tworzy przycisk, przypisuje funkcje obsługi zdarzeń i umieszcza
 * go we właściwym miejscu tabeli pakującej
 */
GtkWidget *UtworzPrzycisk (GtkWidget *tabela,
                           typPrzyciskSapera *pole,
                           int wiersz,
                           int kolumna)
{
    GtkWidget *przycisk;

    /* --- tworzymy przycisk --- */
    przycisk = gtk_toggle_button_new ();
```

```
    /* --- Inicjujemy pola struktury przycisku --- */
    pole->stanPrzycisku = PRZYCISK_NIEZNANY;
    pole->nWiersz = wiersz;
    pole->nKol = kolumna;

    /* --- Musimy sprawdzać, czy kliknięto przycisk --- */
    gtk_signal_connect (GTK_OBJECT (przycisk), "clicked",
                        GTK_SIGNAL_FUNC (kliknieto_plansze), pole);

    /* --- Istotne są także inne zdarzenia --- */
    gtk_signal_connect (GTK_OBJECT (przycisk), "button_press_event",
                        GTK_SIGNAL_FUNC (kliknieto_przycisk), pole);

    /* --- Umieszczamy przycisk we właściwej komórce tabeli --- */
    gtk_table_attach (GTK_TABLE (tabela), przycisk,
                      kolumna, kolumna + 1,
                      wiersz + 1, wiersz + 2,
                      GTK_FILL | GTK_EXPAND,
                      GTK_FILL | GTK_EXPAND,
                      0, 0);

    /* --- Ustawiamy jednolity rozmiar przycisku --- */
    gtk_widget_set_usize (przycisk, SZEROKOSC_PRZYCISKU,
                          WYSOKOSC_PRZYCISKU);

    /* --- Uwidaczniamy przycisk --- */
    gtk_widget_show (przycisk);

    /* --- zwracamy przycisk. --- */
    return (przycisk);
}

/*
 * PoliczSasiednieBomby
 *
 * Oblicza, ile bomb znajduje się w sąsiednich polach
 */
int PoliczSasiednieBomby (int kol, int wiersz)
{
    int i, j;
    int nLiczba = 0;

    /* --- Każde pole oddalone co najwyżej o 1 --- */
```

```
    for (i = MAX (kol-1, 0); i <= MIN (kol+1, nLKolumn-1); i++) {
      for (j = MAX (wiersz-1, 0);
                j <= MIN (wiersz+1, nLWierszy-1); j++) {

        /* --- Jeśli jest w nim bomba... --- */
        if (plansza[i][j].bMaBombe) {

          /* --- ...to zwiększamy licznik --- */
          nLiczba++;
        }
      }
    }
    return (nLiczba);
}

/*
 * UtworzPrzyciskiSapera
 *
 * Tworzy przyciski na planszy Sapera na podstawie tablicy, którą
 * zdefiniowaliśmy na początku programu. Wskaźniki (uchwyty) do
 * przycisków są zachowywane w tej samej tablicy, abyśmy później
 * mieli dostęp do przycisków.
 */
void UtworzPrzyciskiSapera (GtkWidget *tabela,
                            int nKolumnySiatki,
                            int nWierszeSiatki,
                            int bNowePrzyciski)
{
    int ki;
    int wi;
    GtkWidget *przycisk;
    int nBomby;
    typPrzyciskSapera *pole;

    /* --- Uaktualniamy zmienne globalne --- */
    nLKolumn = nKolumnySiatki;
    nLWierszy = nWierszeSiatki;

    bKoniecGry = FALSE;
    bOdPoczatku = TRUE;

    /* --- Uaktualniamy liczbę bomb --- */
    PokazLiczbeBomb ();
```

```
/* --- Sprawdzamy każdy przycisk --- */
for (ki = 0; ki < nLKolumn; ki++) {
   for (wi = 0; wi < nLWierszy; wi++) {

      /* --- Opróżniamy przycisk --- */
      pole = &plansza[ki][wi];
      pole->bMaBombe = 0;
      pole->stanPrzycisku = PRZYCISK_NIEZNANY;

      /* --- Kontrolka już umieszczona na planszy? --- */
      if (bNowePrzyciski) {

        /* --- tworzymy nowy przycisk --- */
        pole->kontrolka = UtworzPrzycisk(tabela, pole, wi, ki);
      } else {

         /* --- wykorzystujemy przycisk ponownie --- */

         /* --- Zwalniamy istniejące rysunki xpm --- */
         ZwolnijPotomka (pole->kontrolka);

         /* --- Wyłączamy przycisk --- */
         gtk_toggle_button_set_state (
                  GTK_TOGGLE_BUTTON (pole->kontrolka),
                  FALSE);
      }
   }
}

/* --- Rozmieszczamy bomby na planszy. --- */
nBomby = nLiczbaBomb;

/* --- Dopóki mamy jeszcze bomby do rozmieszczenia... --- */
while (nBomby > 0) {

   /* --- Obliczamy pozycję rząd/kolumna --- */
   ki = rand () % nLKolumn;
   wi = rand () % nLWierszy;

   /* --- Jeśli bomba jeszcze nie istnieje, tworzymy ją! --- */
   if (plansza[ki][wi].bMaBombe == 0) {
      plansza[ki][wi].bMaBombe = 1;
      nBomby--;
   }
```

```
    }

    /* --- Po rozmieszczeniu bomb obliczamy, ile bomb
     *     przylega do każdego przycisku.
     */

    /* --- Sprawdzamy każdy przycisk --- */
    for (ki = 0; ki < nLKolumn; ki++) {
       for (wi = 0; wi < nLWierszy; wi++) {

          pole = &plansza[ki][wi];

                /* --- Ile przycisków --- */
          pole->nBombyWPoblizu = PoliczSasiednieBomby (ki, wi);
       }
    }
    bOdPoczatku = FALSE;
}

/*
 * UstawSiatke
 *
 * Ustawia siatkę gry na określony rozmiar i liczbę bomb
 */
void UstawSiatke (int nKolumnySiatki, int nWierszeSiatki, int nBomby)
{
    int wiersz, kol;

    /* --- Jeśli istnieje tabela pakująca... --- */
    if (tabela) {

       /* --- ...usuwamy ją wraz z wszystkimi przyciskami --- */
       gtk_widget_destroy (tabela);
    }

    /* --- Tworzymy tabelę na przyciski --- */
    tabela = gtk_table_new (nKolumnySiatki, nWierszeSiatki, FALSE);

    /* --- Dodajemy ją do pionowego pola pakującego --- */
    gtk_box_pack_start (GTK_BOX (ypole), tabela, FALSE, FALSE, 0);

    /* --- Realizujemy tabelę --- */
    gtk_widget_realize (tabela);
```

```
    /* --- Zerujemy grę, określając nowe wartości --- */
    NowaGra (nKolumnySiatki, nWierszeSiatki, nBomby, TRUE);

    /* --- Uwidaczniamy tabelę --- */
    gtk_widget_show (tabela);
}

/*
 * main
 *
 * Tutaj zaczyna się program
 */
int main (int argc, char *argv[])
{
    GtkWidget *okno;
    GdkBitmap *maska;
    GtkStyle  *styl;
    GtkWidget *kontrolka_usmiech;
    GtkWidget *xpole;

    /* --- Inicjacja GTK --- */
    gtk_init (&argc, &argv);

    /* --- Tworzymy okno najwyższego poziomu --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    /* --- Nie pozwalamy na zmianę rozmiarów okna  --- */
    gtk_window_set_policy (GTK_WINDOW (okno), FALSE, FALSE, TRUE);

    /* --- Nadajemy oknu tytuł --- */
    gtk_window_set_title (GTK_WINDOW (okno), "Saper");

    /* --- Zawsze należy pamiętać o podłączeniu sygnału
     *     "delete_event" do głównego okna
     */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);


    ypole = gtk_vbox_new (FALSE, 1);
    gtk_widget_show (ypole);

    /* --- tworzymy menu aplikacji --- */
    UtworzMenu (okno, ypole);
```

```
/* --- Poziome pole na wyniki i przycisk start --- */
xpole = gtk_hbox_new (TRUE, 1);
gtk_widget_show (xpole);

gtk_box_pack_start (GTK_BOX (ypole), xpole, FALSE, FALSE, 0);

/*
 * --- Nie odkryte bomby będziemy wyświetlać na etykiecie
 */

/* --- Dodajemy etykietę z liczbą bomb --- */
etykieta_bomby = gtk_label_new ("");
gtk_box_pack_start (GTK_BOX (xpole), etykieta_bomby,
                    FALSE, FALSE, 0);
gtk_widget_show (etykieta_bomby);

/*
 * --- Tworzymy przycisk startowy z uśmiechniętą twarzą
 */
przycisk_start = gtk_button_new ();

/* --- Musimy wiedzieć, że przycisk został kliknięty --- */
gtk_signal_connect (GTK_OBJECT (przycisk_start),
                    "clicked",
                    GTK_SIGNAL_FUNC (kliknieto_start),
                    NULL);

gtk_box_pack_start (GTK_BOX (xpole), przycisk_start,
                    FALSE, FALSE, 0);
gtk_widget_show (przycisk_start);

/*
 * --- Czas wyświetlamy po prawej
 */

/* --- Dodajemy etykietę z czasem --- */
etykieta_czas = gtk_label_new ("");
gtk_box_pack_start (GTK_BOX (xpole), etykieta_czas,
                    FALSE, FALSE, 0);
gtk_widget_show (etykieta_czas);

/* --- Uwidaczniamy pole --- */
gtk_widget_show (ypole);
```

```
    /* --- Dodajemy pionowe pole do okna aplikacji --- */
    gtk_container_add (GTK_CONTAINER (okno), ypole);
    gtk_widget_show (okno);

   /* --- Tworzymy "uśmiech" i umieszczamy go na przycisku --- */
    UstawIkonePrzyciskuStart ((gchar **) xpm_usmiech);

    /* --- Tworzymy siatkę 10x10. --- */
    UstawSiatke (10, 10, 10);

    /* --- Uruchamiamy pętlę zdarzeń GTK --- */
    gtk_main ();
    exit (0);
}
```

## menu.c

Funkcje w pliku menu.c obsługują tworzenie i wybór elementów menu. Interfejs do kodu sapera ograniczamy do niezbędnego minimum.

```
/*
 * Interfejs GUI aplikacji Sapera.
 *
 * Autor: Eric Harlow
 *
 */

#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
#include <gtk/gtk.h>
#include "rozne.h"

void UstawSiatke (int nLKolumn, int nLWierszy, int nBomby);

/*
 * --- Zmienne globalne
 */
GtkWidget           *glowne_okno;
GtkAccelGroup       *grupa_skrotow;
GtkWidget           *pasek;

/*
```

```
 * menu_Nowy
 *
 * Wywoływana po wybraniu menu Gra
 * Tworzy nową grę
 */
void menu_Nowy (GtkWidget *kontrolka, gpointer dane)
{
    /* --- Parametery nie są używane... --- */

    /* --- Udajemy, że kliknięto przycisk startowy. --- */
    kliknieto_start (NULL, NULL);
}

/*
 * funkcjaPoczatkujacy
 *
 * Wywoływana po wybraniu z menu opcji "Początkujący"
 * Tworzy małą siatkę
 */
void funkcjaPoczatkujacy (GtkWidget *kontrolka, gpointer dane)
{
    if (GTK_CHECK_MENU_ITEM (kontrolka)->active) {
        UstawSiatke (10, 10, 10);
    }
}

/*
 * funkcjaSredni
 *
 * Wywoływana po wybraniu z menu opcji "Średni"
 * Tworzy średnią siatkę
 */
void funkcjaSredni (GtkWidget *kontrolka, gpointer dane)
{
    if (GTK_CHECK_MENU_ITEM (kontrolka)->active) {
        UstawSiatke (20, 15, 40);
    }
}

/*
 * funkcjaZaawansowany
 *
```

```
 * Wywoływana po wybraniu z menu opcji "Zaawansowany"
 * Tworzy największą siatkę, z największą liczbą bomb
 */
void funkcjaZaawansowany (GtkWidget *kontrolka, gpointer dane)
{
   /* --- jeśli element menu jest obecnie aktywny --- */
   if (GTK_CHECK_MENU_ITEM (kontrolka)->active) {

      /* --- Ustawiamy rozmiary siatki --- */
      UstawSiatke (30, 20, 100);
   }
}

/*
 * menu_Zakoncz
 *
 * Wybrano opcję "Zakończ"
 * Zamykamy program
 */
void menu_Zakoncz (GtkWidget *kontrolka, gpointer dane)
{
   gtk_main_quit ();
}

/*
 * menu_OProgramie
 *
 * Wybrano opcję menu "O programie".
 * Pokazujemy informacje o aplikacji.
 */
void menu_OProgramie (GtkWidget *kontrolka, gpointer dane)
{

   PokazOProgramie ();
}

/*
 * UtworzGlowneOkno
 *
 * Tworzy główne okno i związane z nim menu/pasek.
 */
void UtworzMenu (GtkWidget *okno, GtkWidget *ypole)
```

```
{
    GtkWidget *pasekmenu;
    GtkWidget *menu;
    GtkWidget *elmenu;
    GSList *grupa = NULL;

    glowne_okno = okno;

    /* --- tworzymy tablicę skrótów --- */
    grupa_skrotow = gtk_accel_group_new ();
    gtk_accel_group_attach (grupa_skrotow, GTK_OBJECT (okno));

    /* --- Pasek menu --- */
    pasekmenu = gtk_menu_bar_new ();
    gtk_box_pack_start (GTK_BOX (ypole), pasekmenu, FALSE, TRUE, 0);
    gtk_widget_show (pasekmenu);

    /* --------------------
      --- Menu Plik ---
      ---------------- */
    menu = UtworzPodmenuPaska (pasekmenu, "Gra");

    elmenu = UtworzElementMenu (menu, "Nowa", "^N",
                "Nowa gra",
                GTK_SIGNAL_FUNC (menu_Nowy), NULL);

    elmenu = UtworzElementMenu (menu, NULL, NULL,
                NULL, NULL, NULL);

    elmenu = UtworzOpcjonalnyElement (menu, "Początkujący", &grupa,
                GTK_SIGNAL_FUNC (funkcjaPoczatkujacy), NULL);

    elmenu = UtworzOpcjonalnyElement (menu, "Średni", &grupa,
                GTK_SIGNAL_FUNC (funkcjaSredni), NULL);

    elmenu = UtworzOpcjonalnyElement (menu, "Zaawansowany", &grupa,
                GTK_SIGNAL_FUNC (funkcjaZaawansowany), NULL);

    elmenu = UtworzElementMenu (menu, NULL, NULL,
                NULL, NULL, NULL);

    elmenu = UtworzElementMenu (menu, "Zakończ", "",
                "Czy jest bardziej wymowna opcja?",
                GTK_SIGNAL_FUNC (menu_Zakoncz), "zakończ");
```

```
    /* ----------------------
      --- Menu Pomoc ---
      ---------------------- */
    menu = UtworzPodmenuPaska (pasekmenu, "Pomoc");

    elmenu = UtworzElementMenu (menu, "O programie", NULL,
                "O Saperze...",
                GTK_SIGNAL_FUNC (menu_OProgramie),"o programie");
}
```

## Pozostałe pliki

Pozostałe pliki zostały już omówione; można je pobrać spod adresu www.mcp.com (patrz rozdział 1, „Wprowadzenie do GTK+”). Zawierają one funkcje pomocnicze, tworzące menu i okno dialogowe dla opcji „O programie”. Okna dialogowe opisaliśmy w rozdziale 6, „Dalsze kontrolki: ramki, tekst, okna dialogowe, okno wyboru pliku, pasek postępów”, natomiast funkcje tworzące menu opisaliśmy w rozdziale 5, „Menu, paski narzędziowe i podpowiedzi”.

## Podsumowanie

Napisanie programu Sapera powinno pogłębić naszą wiedzę o składaniu prostych aplikacji z pasków narzędziowych, menu i innych kontrolek. Biblioteka GTK+ umożliwia stworzenie wielu typów aplikacji z tych samych kontrolek.