Tworzenie gier 2D i 3D w języku Turbo Pascal

Autor: Piotr Besta
ISBN: 83-7197-680-1
Format: B5, stron: 414

Książka „Tworzenie gier 2D i 3D w języku Turbo Pascal" Piotra Besty skierowana jest do szerokiego grona początkujących programistów, dopiero rozpoczynających swoją wielką przygodę z komputerem. Nie oznacza to, że z książki nie mogą skorzystać doskonale znające Turbo Pascala.

Każdy, kto kiedykolwiek grał w gry komputerowe, zapragnął z pewnością stworzyć własną, idealną grę. Dzięki tej książce osoby te będą mogły spełnić swoje marzenia w profesjonalny sposób.

Rozpoczynając czytanie książki nie musisz podsiać doświadczenia w pisaniu programów, wszystkiego dowiesz się, czytając kolejne rozdziały, a zgromadzona wiedza oraz doświadczenie pozwoli na szybkie rozpoczęcie samodzielnych prac nad dowolną grą. Książka w szczegółowy sposób wyjaśnia proces i sposoby wyświetlania grafiki w grach, tajemnice oraz sztuczki w nich zastosowane.

- Poznasz techniki wprawiania w ruch grafiki dwu- i trójwymiarowej.
- Nauczysz się tworzyć grafikę spotykaną w grach typu RTS i FPP.
- Poznasz podstawy szybkiego języka programowania, jakim jest Assembler.

A po ukończeniu książki śmiało będziesz mógł przystąpić do tworzenia takich gier jak Settlers, Quake, Dune, StarCraft, Descent i im podobnych. Musisz jednak wykazać się niebywałą determinacją i poświęceniem – i tego między innymi nauczy cię ta książka.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 6.

Komunikacja z graczem i wyświetlanie tekstu

W tym rozdziale omówimy dwa sposoby komunikowania się gier z graczem. Pierwszy sposób dotyczyć będzie klawiatury; poznamy jej działanie, nauczymy się odczytywać informację mówiącą o tym, który klawisz jest wciskany, a który zwalniany, w końcu napiszemy przykładowy program obsługujący klawiaturę.

Drugi rodzaj komunikacji związany będzie z myszą; nauczymy się wyświetlać kursor myszy, odczytywać jego położenie i pobierać informacje o stanie przycisków myszy. Paragraf poświęcony myszy również zakończymy demonstracyjnym programem, w którym wykorzystamy zdobytą wiedzę w praktyczny sposób.

Osobną część rozdziału stanowi podrozdział poświęcony technikom wyświetlania tekstu w trybie 13h. Stworzymy własną czcionkę i nauczymy się ją stosować. Na koniec napiszemy wygodną procedurę wypisującą tekst, podobną trochę do procedury `Writeln`.

Bierzmy się do pracy!

Klawiatura

Klawiatura jest zaraz obok myszy jednym z najintensywniej wykorzystywanych urządzeń dołączonych do komputera. Korzystamy z niej podczas tworzenia programów, prac domowych, ściąg i wielu innych tekstów. Nas jednak interesuje wyłącznie jej zastosowanie w grach komputerowych. Rysunek. 6.1 przedstawia standardową klawiaturę.

Rysunek 6.1. *Standardowa klawiatura*

Jako programiści gier chcemy mieć możliwość sprawdzenia, czy dany klawisz jest naciśnięty, nie naciśnięty czy może przytrzymany.

Zacznijmy od zrozumienia sposobu działania klawiatury. Z klawiaturą połączony jest tzw. kontroler klawiatury (dokładniej jeden z układów scalonych: 8042, 8741 lub 8742). Możemy odnosić się do niego za pomocą dwóch portów: portu wyjściowego — 60h i portu sterującego — 64h. Są to porty dwukierunkowe, co oznacza, że możemy zarówno odczytywać z nich dane, jak i je do nich zapisywać. Odczytując wartość z portu 64h, otrzymujemy trochę informacji o stanie klawiatury.

Oto opis bajtu pobranego z portu 64h.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7 — błąd parzystości (wyłącznie w modelach PS/2):

1 = ostatni kod przesłano z błędem,

0 = ostatni kod przesłano bez błędu;

bit 6 — czas oczekiwania na nadesłanie danych:

1 = przekroczony,

0 = w normie;

bit 5 — stan portu wyjściowego klawiatury:

1 = port zawiera dane od jednostki dodatkowej (wyłącznie w PS/2),

0 = port zawiera dane od klawiatury;

bit 4 — stan blokady klawiatury:

1 = klawiatura odblokowana,

0 = klawiatura zablokowana;

bit 3 — zawartość przesyłanego bajtu:

1 = bajt rozkazowy wysłany przez port 64h,

0 = bajt danych wysłany przez port 60h;

bit 2 — wynik przeprowadzenia autotestu klawiatury:

1 = test przebiegł bezbłędnie,

0 = błąd;

bit 1 — stan portu 60h (zapis):

1 = dane przesyłane przez procesor znajdują się jeszcze w porcie,

0 = port 60h jest pusty;

bit 0 — stan portu 60h (odczyt):

1 = dane przesyłane przez kontroler klawiatury znajdują się jeszcze w porcie 60h,

0 = port 60h jest pusty.

Wymieniona przedtem dodatkowa jednostka oznacza mysz, która w systemach PS/2 jest obsługiwana przez kontroler klawiatury.

Przeanalizujmy krok po kroku zdarzenie związane z naciśnięciem i zwalnianiem klawisza. Klawiatura posługuje się tzw. kodami skaningowymi klawiszy. Różnią się one od kodów ASCII numeracją, na przykład w kodzie ASCII klawiszowi *a* odpowiada liczba 97, natomiast w kodzie skaningowym odpowiada mu liczba 30. Kod skaningowy w odróżnieniu od ASCII nie wyróżnia wielkich i małych liter, wielkość liter jest rozstrzygana w oparciu o stan klawiszy *Caps Lock* i *Shift*.

W momencie naciśnięcia klawisza do kontrolera klawiatury trafia jego kod skaningowy, po czym zostaje wywołane przerwanie sprzętowe o numerze 09h (tzw. przerwanie klawiatury). W tym momencie procesor przerywa pracę i wykonuje procedurę obsługującą przerwanie klawiatury; odczytuje ona kod skaningowy naciśniętego klawisza, przekształca go na kod ASCII i umieszcza w specjalnym buforze BIOS-u. Z tego buforu korzysta m.in. standardowa funkcja Turbo Pascala `ReadKey`. My jednak nie będziemy tego robić, ponieważ potrzebujemy jak najszybszego dostępu do klawiatury. W informacje o stanie poszczególnych klawiszy będziemy zaopatrywać się bezpośrednio u kontrolera klawiatury, odczytując zawartość portu 60h.

Naciśnięcie dowolnego klawisza powoduje umieszczenie w porcie 60h jego kodu skaningowego, zwolnienie klawisza również powoduje umieszczenie w porcie 60h jego kodu skaningowego, z tą różnicą, że jest on zwiększony o liczbę 128 (kody skaningowe klawiszy zamieszczono w dodatku na końcu książki).

Załóżmy, że wciskamy klawisz *C*, do portu 60h trafia liczba 46 (kod skaningowy klawisza *C*); gdy zwolnimy klawisz *C*, do portu 60h trafi liczba 174 (46+128).

Występowanie różnych kodów dla wciskanych i zwalnianych klawiszy wykorzystać możemy do wykrycia zdarzenia przytrzymania przez gracza danego klawisza. Jest to całkiem proste. Gdy odbierzemy kod skaningowy oznaczający naciśnięcie na przykład klawisza *C*, traktować będziemy go jak klawisz wciśnięty, dopóki nie odbierzemy z portu 60h liczby 174, oznaczającej jego zwolnienie.

Jak odczytać wartość z portu 60h? Można to zrobić choćby tak:

```
var
wartosc : byte;
.
.
.
wartosc := port[$60];
```

Musimy jednak pamiętać o tym, że podczas naciskania i zwalniania klawiszy generowane jest przerwanie klawiatury; powoduje to, że procesor przerywa pracę i wykonuje procedurę obsługującą przerwanie klawiatury, a ona odczytuje kod skaningowy

klawisza z portu 60h. Tłumacząc to w prostszy sposób, możemy powiedzieć, że zostajemy odcięci od portu 60h przez przerwanie klawiatury, ponieważ jest ono wykonywane niezależnie od nas. Jedyne, co możemy zrobić, to zakazać systemowi obsługi przerwania sprzętowego 09h (numer przerwania klawiatury) i samemu przejąć kontrolę nad klawiaturą.

O wykonywaniu przez system poszczególnych przerwań decyduje zawartość portu 21h; jest to tzw. port maski przerwań. Każdy bit znajdującego się w nim bajtu decyduje o wykonywaniu jednego przerwania. Jeżeli bit jest wyzerowany, to dane przerwanie jest obsługiwane przez system, natomiast jeżeli bit jest ustawiony, to dane przerwanie nie jest wykonywane. Oto wyjaśnienie znaczenia poszczególnych bitów portu 21h.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7 — przerwanie portu równoległego:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 6 — przerwanie sterownika stacji dyskietek:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 5 — przerwanie sterownika dysku twardego:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 4 — przerwanie portu szeregowego nr 1:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 3 — przerwanie portu szeregowego nr 2:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 2 — przerwanie drugiego układu 8259:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 1 — przerwanie klawiatury:

1 = nie jest obsługiwane,

0 = jest obsługiwane;

bit 0 — przerwanie układu czasowego:

1 = nie jest obsługiwane,

0 = jest obsługiwane.

Tak więc aby uniemożliwić przekazanie sterowania do procedury obsługującej przerwanie klawiatury, należy ustawić pierwszy bit portu 21h. Można to zrobić w następujący sposób:

```
port[$21] := $02;        {02h = 00000010b}
```

Po tej operacji klawiatura przestanie reagować. Gdy odłączymy przerwanie klawiatury, musimy pamiętać o tym, aby przed wyłączeniem przez gracza programu (gry) przyłączyć je z powrotem, w przeciwnym razie użytkownik nie będzie mógł normalnie kontynuować pracy w systemie. Dokonujemy tego, zerując pierwszy bit portu 21h.

Pora na naukę odczytywania stanu klawiszy. Jest to całkiem proste, wystarczy, że będziemy bezustannie pobierać i sprawdzać wartość z portu 60h.

Napiszmy procedurę, która będzie otrzymywać poprzez parametr zmienną typu `byte`; jeżeli w porcie 60h znajdzie informację od klawiatury dotyczącą naciśnięcia lub zwolnienia klawisza, to zapisze ją do tej zmiennej. W sytuacji gdy port 60h będzie pusty, umieści w zmiennej wartość 0. Procedurę nazwiemy `g_pobierz_skan_kod`, a oto jej kod:

```
procedure g_pobierz_skan_kod(var kod_scan : byte);
var
temp : byte;
begin

{Pobranie informacji o klawiaturze.}
temp := port[$64];

{Sprawdzenie, czy bufor nie jest pusty i czy bajt znajdujący się
 w porcie 60h pochodzi od klawiatury. Jeżeli któryś z tych dwóch
 warunków nie jest spełniony, odczytujemy z portu 60h wartość, aby go
 opróżnić, co pozwoli klawiaturze na zapisanie do niego kolejnych
 informacji; zapisujemy zero do otrzymanej poprzez parametr
 zmiennej i  wychodzimy z procedury.}
if ((temp and $01) = 0) then
                        begin
                        kod_scan := 0;
                        exit;
                        end;

if ((temp and $20) = 32) then
                        begin
                        temp := port[$60];
                        kod_scan := 0;
                        exit;
                        end;

{Pobierz kod scan naciśniętego lub zwolnionego klawisza.}
kod_scan := port[$60];

end; {koniec procedury g_pobierz_skan_kod}
```

Z pewnością zastanawiasz się, dlaczego wartość z portu 60h jest odczytywana w sytuacji, gdy pochodzi od jednostki dodatkowej; jest tak, gdyż klawiatura nie może zapisać do portu 60h nowej wartości, dopóki procesor go nie opróżni.

Kolejny problem, jaki musimy rozwiązać, dotyczy automatycznego powtarzania kodu klawiszy. Gdy przytrzymamy na moment jakikolwiek klawisz, klawiatura zaczyna zapisywać do portu 60h od 2 do 30 razy w ciągu sekundy kod tego klawisza, co jak nietrudno wywnioskować, powoduje zapchanie portu 60h. Z tym problemem uporać możemy się tylko w jeden sposób, a mianowicie zmniejszając liczbę powtórzeń zapisu do portu 60h kodu przytrzymanego klawisza. Do zmiany tej właściwości klawiatury służy funkcja 03h przerwania 16h.

Funkcja 03h przerwania 16h powoduje ustalenie nowej częstotliwości powtarzania znaków oraz opóźnienia czasowego, po upływie którego ma się ono zacząć. Parametry wejściowe są następujące:

- AH — numer funkcji przerwania 16h, z której chcemy skorzystać, oczywiście będzie to 03h;
- BL — prędkość powtarzania znaków przez klawiaturę, wyrażona w znakach na sekundę; możliwości ustawień jest bardzo wiele:

```
00h = 30, 10h = 7.5,
01h = 26.7, 11h = 6.7,
02h = 24, 12h = 6,
03h = 21.8, 13h = 5.5,
04h = 20, 14h = 5,
05h = 18.5, 15h = 4.6,
06h = 17.1, 16h = 4.3,
07h = 16, 17h = 4,
08h = 15, 18h = 3.7,
09h = 13.3, 19h = 3.3,
0Ah = 12, 1Ah = 3,
0Bh = 10.9, 1Bh = 2.7,
0Ch = 10, 1Ch = 2.5,
0Dh = 9.2,, 1Dh = 2.3,
0Eh = 8.5, 1Eh = 2.1,
0Fh = 8, 1Fh = 2;
```

- BH — opóźnienie zadziałania powtarzania klawiszy:

```
00h = 0.25 sek.
01h = 0.5 sek.
10h = 0.75 sek.
11h = 1.0 sek.
```

Napiszmy krótką procedurę, która pozwoli nam ustalać dowolną częstotliwość powtarzania kodu klawiszy i czas opóźniający jego zadziałanie. Procedurę nazwijmy g_powtarzanie_klawiszy, a oto jak powinien wyglądać jej kod:

```
procedure g_powtarzanie_klawiszy( liczba, opoznienie : byte);
begin
asm

mov ah, 03h        {Ładuj 03h do AH, numer funkcji przerwania 16h.}
mov bl, liczba     {Ładuj liczba do BL, liczba powtórzeń kodu klawiszy.}
```

```
mov bh, opoznienie {Ładuj opóźnienie do BH.}
int 16h            {Wywołaj przerwanie 16h.}

end;
end; {koniec procedury g_powtarzanie_klawiszy}
```

Procedura otrzymuje dwa parametry, w pierwszym określamy liczbę powtórzeń kodu klawisza w ciągu sekundy, a w drugim czas opóźniający rozpoczęcie jego działania.

Procedury `g_pobierz_skan_kod` oraz `g_powtarzanie_klawiszy` dołączamy do modułu `graf_fx`. Zanim przejdziemy do napisania przykładowego programu, stwórzmy procedurę, która będzie odłączała i przyłączała klawiaturę. Przekazujemy do niej wartość typu `Boolean`; jeżeli będzie to `TRUE`, procedura zezwoli systemowi na obsługę przerwania klawiatury; w wypadku przekazania wartości `FALSE` procedura odcina system od przerwania klawiatury. Procedurę nazwiemy `g_sys_klawiatura`. Oto jej kod:

```
procedure g_sys_klawiatura( stan : boolean);
begin

if(stan = TRUE) then
   port[$21] := $00    {Włączenie obsługi przerwania klawiatury.}
   else
   port[$21] := $02;   {Wyłączenie obsługi przerwania klawiatury.}

end; {koniec procedury g_sys_klawiatura}
```

Procedurę `g_sys_klawiatura` dołączamy do modułu `graf_fx`. Program z listingu 6.1 przejmuje kontrolę nad klawiaturą, ustala częstotliwość powtarzania kodu klawiszy na 2 w ciągu sekundy, odczytuje zawartość z portu 60h i wyświetla ją na ekranie.

Listing 6.1. *Klawiatura (p_6_1.pas)*

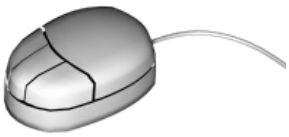
```
program p_6_1;

{// D O Ł Ą C Z O N E   M O D U Ł Y
{////////////////////////////////////////}
uses crt,graf_fx;

{// Z M I E N N E   G L O B A L N E
{////////////////////////////////////////}
var
kod_skan : byte;

{// G Ł Ó W N Y   P R O G R A M
{////////////////////////////////////////}

begin
{Wyczyszczenie ekranu.}
clrscr;

{Ustawiamy częstotliwość powtarzania klawiszy na 2 w ciągu sekundy,
 ustalamy opóźnienie rozpoczęcia powtarzania na 0.5 s oraz
 zakazujemy systemowi obsługi przerwania klawiatury.}
g_powtarzanie_klawiszy($1f, $01);
```

```
g_sys_klawiatura(FALSE);

{Wykonująca się w nieskończoność pętla while do.}
while (1 = 1) do
begin

{Pobieramy wartość z portu 60h.}
g_pobierz_skan_kod(kod_skan);

{Jeżeli pobrany kod odpowiada klawiszowi ESC, przerywamy pętlę.}
if (kod_skan = 1) then break;

{Jeżeli kod należy do dowolnego innego klawisza, to go wypisujemy.}
if (kod_skan <> 0) then writeln(kod_skan);

end; {koniec pętli while do}

{Zezwalamy systemowi na obsługę przerwania klawiatury.}
g_sys_klawiatura(TRUE);

end. {koniec programu}
```

Mysz

W dzisiejszych systemach operacyjnych mających rozbudowany interfejs graficzny mysz jest niezastąpionym narzędziem pracy. To prawda, że po ikonach i folderach poruszać możemy się również wyłącznie za pomocą klawiatury, jednak dzięki myszy jest to dużo prostsze. Odnosząc się do gier komputerowych, musimy stwierdzić, że nie jest możliwe wybranie jednego urządzenia kontrolującego; po prostu jedne gry napisane są w taki sposób, że korzystają wyłącznie z klawiatury, inne wyłącznie z myszy, a jeszcze inne z obydwu tych urządzeń naraz. Na rysunku 6.2 przedstawiono standardową mysz.

Rysunek 6.2.
Standardowa mysz



Obsługa myszy od strony programowej jest dużo łatwiejsza niż obsługa klawiatury. Odbywa się ona poprzez przerwanie 33h. Oto opis funkcji przerwania 33h; z reguły jest ich ponad 30, my wykorzystywać będziemy tylko kilka z nich.

```
00h – Inicjalizacja myszy
 wywołanie:
 AX = 0000h
 Powrót:
 AX =
 0000h - mysz zainstalowana
 FFFFh – brak myszy
 BX – liczba przycisków
01h – Wyświetlenie kursora myszy
```

```
 wywołanie:
 AX = 0001h

02h – Ukrycie kursora myszy
 wywołanie:
 AX = 0002h

03h – Status i położenie myszy
 wywołanie:
 AX = 0003h
 Powrót:
 BX =
 jeżeli bit 0 ustawiony to lewy przycisk wciśnięty
 jeżeli bit 1 ustawiony to prawy przycisk wciśnięty
 jeżeli bit 2 ustawiony to środkowy przycisk wciśnięty
 CX – położenie poziome(w trybie 13h od 0 do 639, tak właśnie jest!)
 DX – położenie pionowe(w trybie 13h od 0 do 199)
```

Napiszmy teraz dwie procedury, które wykorzystamy do obsługi myszy. Pierwsza będzie wyświetlać i chować kursor. Nazwijmy ją `g_pokaz_kursor`; oto jej kod:

```
procedure g_pokaz_kursor( status : boolean);
begin

if (status = TRUE) then
                   asm
                   mov ax, 0001h  {Ładuj 0001h do AX.}
                   int 33h        {Wywołaj przerwanie 33h.}
                   end else
                   asm
                   mov ax, 0002h  {Ładuj 0002h do AX.}
                   int 33h        {Wywołaj przerwanie 33h.}
                   end;

end; {koniec procedury g_pokaz_kursor}
```

Druga procedura pobierać będzie informacje o stanie przycisków myszy i położeniu kursora. Pobrane dane zostaną zapisane w pięciu zmiennych, przekazywanych do procedury poprzez parametry. Pierwsze trzy dotyczą stanu lewego, środkowego i prawego przycisku myszy, pozostałe dwa to zmienne, w których zostanie umieszczona pozycja kursora myszy. Procedurę nazywamy `g_pobierz_status_myszy`, a oto jej kod:

```
procedure g_pobierz_status_myszy(var lewy, srodkowy, prawy : boolean;
                                 var poz_x, poz_y : word);
var
temp : word;
temp_poz_x, temp_poz_y : word;
begin

asm
mov ax, 0003h                  {Ładuj 00003h do AX.}
int 33h                        {Wywołaj przerwanie 33h.}
mov word ptr temp, bx          {Ładuj BX do temp.}
mov word ptr temp_poz_x, cx    {Ładuj CX do temp_poz_x.}
mov word ptr temp_poz_y, dx    {Ładuj DX do temp_poz_y.}
end;
poz_x := temp_poz_x;
poz_y := temp_poz_y;
```

```
if ((temp and $01) <> 0) then lewy := TRUE else lewy := FALSE;
if ((temp and $02) <> 0) then prawy := TRUE else prawy := FALSE;
if ((temp and $04) <> 0) then srodkowy := TRUE else srodkowy := FALSE;

end; {koniec procedury g_pobierz_status_myszy}
```

Wyjaśnijmy, w jaki sposób można zmienić kształt kursora. Po pierwsze, nie wyświetlamy standardowego kursora myszy, czyli nie używamy procedury `g_pokaz_kursor` z parametrem `TRUE`. Tworzymy bitmapę przedstawiającą nasz wymarzony wskaźnik, podczas pracy programu odczytujemy pozycję kursora myszy i we wskazanym miejscu wyświetlamy obraz kursora pochodzący z bitmapy na przykład za pomocą procedury `g_wyswietl_obraz_13h_k`.

Gdy używamy własnego kursora, musimy narysować go za każdym razem, gdy tworzymy nową klatkę obrazu. Z reguły jego rysowanie odbywa się na samym końcu, już po wyświetleniu wszystkich obiektów. Gdy natomiast używamy standardowego kursora wyświetlanego przez kartę graficzną, niczym nie musimy się przejmować; karta graficzna odpowiada za to, by kursor został narysowany w odpowiedniej kolejności i w odpowiednim miejscu.

Program z listingu 6.2 demonstruje działanie naszych procedur obsługujących mysz. W programie poruszamy standardowym kursorem; w momencie naciśnięcia lewego przycisku myszy na ekranie zostaje narysowany piksel o losowo wybranym kolorze. Naciśnięcie prawego przycisku myszy kończy działanie programu.

Listing 6.2. *Mysz (p_6_2.pas)*

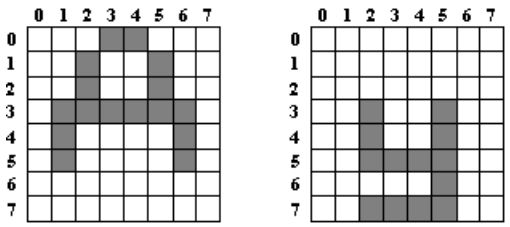
```
program p_6_2;

{// D O Ł Ą C Z O N E   M O D U Ł Y
{////////////////////////////////////////}
uses graf_fx;

{// Z M I E N N E   G L O B A L N E
{////////////////////////////////////////}
var
lewy, srodkowy, prawy : boolean;
m_poz_x, m_poz_y : word;

{// G Ł Ó W N Y   P R O G R A M
{////////////////////////////////////////}

begin

{Włączamy tryb graficzny 320×200×256.}
g_ustaw_tryb_vga($13);

{Wyświetlamy kursor.}
g_pokaz_kursor(TRUE);
while (1 = 1) do
begin
```

```
{Odczytanie danych o stanie myszy.}
g_pobierz_status_myszy(lewy, srodkowy, prawy, m_poz_x, m_poz_y);

{Jeżeli lewy przycisk jest wciśnięty, rysujemy piksel w pozycji
 kursora; wartość określającą poziome położenie myszy dzielimy przez
 2, ponieważ należy ona do przedziału od 0 do 639, ekran ma natomiast
 szerokość 320 pikseli.}
if(lewy = TRUE) then g_rysuj_piksel_13h(m_poz_x div 2, m_poz_y,
                                        random(256));

{Jeżeli prawy przycisk jest wciśnięty, chowamy kursor i przerywamy pętlę.}
if(prawy = TRUE) then
                 begin
                 g_pokaz_kursor(FALSE);
                 break;
                 end;

end; {koniec pętli while do}

end. {koniec programu}
```

Wyświetlanie tekstu

Omawianie sposobów wyświetlania tekstu rozpoczęliśmy już w pierwszym rozdziale; w trybie tekstowym korzystaliśmy z procedur `Writeln` i `Write`, pierwsza wyświetla podany tekst i przenosi kursor do następnej linii, druga wyświetla tekst bez przenoszenia kursora. W trybie graficznym obsługiwanym przez sterowniki BGI stosowaliśmy procedurę `OuttexXY`, która wyświetla podany tekst w określonym miejscu.

Pracując w trybie graficznym 13h, nie mamy do dyspozycji procedur wypisujących tekst, musimy polegać wyłącznie na sobie i samodzielnie je stworzyć. W pierwszej kolejności musimy zająć się utworzeniem czcionki. Każdy jej znak będzie miał rozmiar 8×8 pikseli. Spójrz na rysunek 6.3 przedstawiający litery *A* i *y*.



Rysunek 6.3. *Sposób budowy znaków za pomocą pikseli*

W skład czcionki wchodzić będą 72 znaki:

- spacja,
- kropka,
- wielkie litery: *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I*, *J*, *K*, *L*, *M*, *N*, *O*, *P*, *Q*, *R*, *S*, *T*, *U*, *V*, *W*, *X*, *Y*, *Z*,
- małe litery: *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*, *k*, *l*, *m*, *n*, *o*, *p*, *q*, *r*, *s*, *t*, *u*, *v*, *w*, *x*, *y*, *z*,

- cyfry: *0*, *1*, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*,
- znaki specjalne: -, +, *‘*, *’*, */*, *, :, =, _.

Do reprezentacji każdego znaku użyjemy 8 bajtów. Jak to możliwe? Przecież każda litera składa się z 64 pikseli! Masz rację. Jednak zauważ, że 8 bajtów to zespół 64 bitów. Do opisu każdego piksela użyjemy 1 bitu. Jeżeli będzie on ustawiony, oznaczać to będzie, że należy go wyświetlić; jeżeli będzie wyzerowany —narysowanie piksela nie nastąpi. Na rysunku 6.4 znajdują się litery *A* i *y*, przedstawione za pomocą bitów.

Rysunek 6.4. *Reprezentacja znaków za pomocą bitów*

bajty:	bity: 7	6	5	4	3	2	1	0	zapis szesnastkowy:
1	0	0	0	1	1	0	0	0	18h
2	0	0	1	0	0	1	0	0	24h
3	0	0	1	0	0	1	0	0	24h
4	0	1	1	1	1	1	1	0	7Eh
5	0	1	0	0	0	0	1	0	42h
6	0	1	0	0	0	0	1	0	42h
7	0	0	0	0	0	0	0	0	00h
8	0	0	0	0	0	0	0	0	00h

	7	6	5	4	3	2	1	0	zapis szesnastkowy:
1	0	0	0	0	0	0	0	0	00h
2	0	0	0	0	0	0	0	0	00h
3	0	0	0	0	0	0	0	0	00h
4	0	0	1	0	0	1	0	0	24h
5	0	0	1	0	0	1	0	0	24h
6	0	0	1	1	1	1	0	0	3Ch
7	0	0	0	0	0	1	0	0	04h
8	0	0	1	1	1	1	0	0	3Ch

Zanim przystąpimy do budowy procedury wypisującej tekst, musimy utworzyć zmienną, która będzie przechowywała wszystkie wzorce znaków. Posłużymy się w tym celu tablicą złożoną z 71 elementów typu `wzorzec_znaku` (tablica złożona jest z 71, a nie 72 elementów, ponieważ nie bierzemy pod uwagę znaku spacji; po jego napotkaniu przejdziemy do rozpatrywania kolejnego znaku). Definicję typu `wzorzec_znaku` tworzymy w następujący sposób:

```
type wzorzec_znaku = array[0..7] of byte;
```

Elementom tablicy nadamy wartość już w momencie ich definiowania. W tym celu należy dodać przed nazwą tablicy dyrektywę `const` (tzn. stała). Zmienne poprzedzone dyrektywą `const` mają wszystkie właściwości zwykłych zmiennych (zdefiniowanych bez słowa `const`), różni je jedynie możliwość nadania im początkowej wartości. Zmienne określone jako `const` najlepiej definiować po definicji nowych typów danych. Definicja tablicy zawierającej wzorce znaków naszej czcionki wygląda następująco:

```
const g_tab_znakow : array[0..70] of wzorzec_znaku =
      (
       ($00, $00, $00, $00, $00, $08, $00, $00),  {0}{'.'}
       ($18, $24, $24, $24, $24, $18, $00, $00),  {1}{'0'}
       ($18, $28, $08, $08, $08, $3C, $00, $00),  {2}{'1'}
       ($38, $44, $08, $10, $20, $7C, $00, $00),  {3}{'2'}
       ($38, $44, $04, $18, $44, $38, $00, $00),  {4}{'3'}
       ($20, $20, $24, $3E, $04, $04, $00, $00),  {5}{'4'}
       ($3E, $20, $3C, $02, $02, $3C, $00, $00),  {6}{'5'}
       ($1C, $20, $38, $24, $24, $18, $00, $00),  {7}{'6'}
       ($3C, $44, $08, $10, $10, $10, $00, $00),  {8}{'7'}
       ($18, $24, $18, $24, $24, $18, $00, $00),  {9}{'8'}
       ($18, $24, $1C, $04, $04, $18, $00, $00),  {10}{'9'}
       ($18, $24, $24, $7E, $42, $42, $00, $00),  {11}{'A'}
       ($7C, $22, $3C, $22, $22, $7C, $00, $00),  {12}{'B'}
       ($3C, $42, $40, $40, $42, $3C, $00, $00),  {13}{'C'}
       ($7C, $22, $22, $22, $22, $7C, $00, $00),  {14}{'D'}
       ($7E, $22, $38, $20, $22, $7E, $00, $00),  {15}{'E'}
       ($7E, $22, $38, $20, $20, $70, $00, $00),  {16}{'F'}
```

```
  ($1C, $22, $20, $26, $22, $1C, $00, $00),  {17}{'G'}
  ($44, $44, $7C, $44, $44, $44, $00, $00),  {18}{'H'}
  ($38, $10, $10, $10, $10, $38, $00, $00),  {19}{'I'}
  ($70, $10, $10, $90, $90, $70, $00, $00),  {20}{'J'}
  ($64, $28, $30, $28, $24, $64, $00, $00),  {21}{'K'}
  ($38, $10, $10, $10, $12, $1E, $00, $00),  {22}{'L'}
  ($22, $36, $2A, $22, $22, $22, $00, $00),  {23}{'M'}
  ($22, $32, $2A, $26, $22, $22, $00, $00),  {24}{'N'}
  ($38, $44, $44, $44, $44, $38, $00, $00),  {25}{'O'}
  ($7C, $22, $3C, $20, $20, $70, $00, $00),  {26}{'P'}
  ($38, $44, $44, $44, $4C, $38, $08, $04),  {27}{'Q'}
  ($7C, $22, $3C, $22, $22, $62, $00, $00),  {28}{'R'}
  ($18, $24, $10, $08, $24, $18, $00, $00),  {29}{'S'}
  ($7F, $49, $08, $08, $08, $1C, $00, $00),  {30}{'T'}
  ($22, $22, $22, $22, $22, $1C, $00, $00),  {31}{'U'}
  ($22, $22, $22, $22, $14, $08, $00, $00),  {32}{'V'}
  ($22, $22, $22, $2A, $36, $22, $00, $00),  {33}{'W'}
  ($22, $14, $08, $14, $22, $22, $00, $00),  {34}{'X'}
  ($22, $22, $14, $08, $08, $1C, $00, $00),  {35}{'Y'}
  ($3E, $22, $04, $08, $12, $3E, $00, $00),  {36}{'Z'}
  ($00, $00, $38, $44, $44, $3E, $00, $00),  {37}{'a'}
  ($20, $20, $3C, $22, $22, $3C, $00, $00),  {38}{'b'}
  ($00, $00, $1C, $20, $20, $1C, $00, $00),  {39}{'c'}
  ($02, $02, $1E, $22, $22, $1E, $00, $00),  {40}{'d'}
  ($00, $1C, $22, $3E, $20, $1C, $00, $00),  {41}{'e'}
  ($18, $24, $20, $70, $20, $20, $00, $00),  {42}{'f'}
  ($00, $00, $1C, $22, $22, $1E, $02, $1C),  {43}{'g'}
  ($20, $20, $3C, $22, $22, $22, $00, $00),  {44}{'h'}
  ($00, $10, $00, $10, $10, $10, $00, $00),  {45}{'i'}
  ($00, $10, $00, $10, $10, $10, $10, $70),  {46}{'j'}
  ($20, $20, $24, $38, $24, $24, $00, $00),  {47}{'k'}
  ($20, $20, $20, $20, $20, $38, $00, $00),  {48}{'l'}
  ($00, $00, $63, $55, $49, $41, $00, $00),  {49}{'m'}
  ($00, $00, $2E, $11, $11, $11, $00, $00),  {50}{'n'}
  ($00, $00, $1C, $22, $22, $1C, $00, $00),  {51}{'o'}
  ($00, $00, $3C, $22, $22, $3C, $20, $20),  {52}{'p'}
  ($00, $00, $1E, $22, $22, $1E, $02, $02),  {53}{'q'}
  ($00, $00, $2C, $12, $10, $10, $00, $00),  {54}{'r'}
  ($00, $3C, $22, $18, $44, $38, $00, $00),  {55}{'s'}
  ($10, $10, $38, $10, $10, $10, $00, $00),  {56}{'t'}
  ($00, $00, $24, $24, $24, $18, $00, $00),  {57}{'u'}
  ($00, $00, $14, $14, $14, $08, $00, $00),  {58}{'v'}
  ($00, $00, $41, $49, $55, $63, $00, $00),  {59}{'w'}
  ($00, $00, $24, $18, $18, $24, $00, $00),  {60}{'x'}
  ($00, $00, $24, $24, $24, $3C, $04, $3C),  {61}{'y'}
  ($00, $00, $3C, $08, $10, $3C, $00, $00),  {62}{'z'}
  ($00, $00, $00, $3C, $00, $00, $00, $00),  {63}{'-'}
  ($00, $00, $10, $38, $10, $00, $00, $00),  {64}{'+'}
  ($00, $00, $00, $00, $00, $10, $10, $20),  {65}{','}
  ($00, $02, $04, $08, $10, $20, $00, $00),  {66}{'/'}
  ($00, $14, $08, $14, $00, $00, $00, $00),  {67}{'*'}
  ($00, $00, $08, $00, $00, $08, $00, $00),  {68}{':'}
  ($00, $00, $3C, $00, $3C, $00, $00, $00)   {69}{'='}
  ($00, $00, $00, $00, $00, $7E, $00, $00)   {70}{'_'}
);
```

Definicje tablicy `g_tab_znakow` oraz typu danych `wzorzec_znaku` dodajemy do modułu `graf_fx`. Możemy teraz przystąpić do napisania i omówienia procedury rysującej tekst; nazwiemy ją po prostu `g_wyswietl_tekst_13h`. Oto jej kod:

```
procedure g_wyswietl_tekst_13h( wsk_buforu : pointer; x, y : word; tekst : string;
kolor : byte);
var
liczba_znakow_w_tekscie : byte;
licznik_znakow,licznik_bitow : byte;
licznik_bajtow : byte;
indeks : byte;
przesuniecie : word;
maska_bitowa : byte;
begin

{Jeżeli tekst leży za nisko, przerywamy pracę procedury.}
if((y + 8) > 199) then exit;

{Obliczmy przesunięcie w buforze do pierwszego piksela tekstu.}
przesuniecie := (y * 320) + x;

{Pobieramy liczbę znaków w tekście.}
liczba_znakow_w_tekscie := ord(tekst[0]);

{Rysujemy wszystkie znaki.}
for licznik_znakow := 1 to liczba_znakow_w_tekscie do
begin

{Jeżeli napotykamy spację, przechodzimy do kolejnego znaku.}
if(tekst[licznik_znakow] = ' ') then
begin
x := x + 8;
if (x > 312) then break;
przesuniecie := (y * 320) + x;
continue;
end else {W przeciwnym przypadku sprawdzamy, czy znak jest kropką
          lub innym znakiem specjalnym.}

if(tekst[licznik_znakow] = '.') then indeks := 0 else
if(tekst[licznik_znakow] = '-') then indeks := 63 else
if(tekst[licznik_znakow] = '+') then indeks := 64 else
if(tekst[licznik_znakow] = ',') then indeks := 65 else
if(tekst[licznik_znakow] = '/') then indeks := 66 else
if(tekst[licznik_znakow] = '*') then indeks := 67 else
if(tekst[licznik_znakow] = ':') then indeks := 68 else
if(tekst[licznik_znakow] = '=') then indeks := 69 else
if(tekst[licznik_znakow] = '_') then indeks := 70 else

{Jeśli nią nie jest, sprawdzamy, czy jest cyfrą.}
if((tekst[licznik_znakow] >= '0') and
   (tekst[licznik_znakow] <= '9')) then
   indeks := ord(tekst[licznik_znakow]) - 47 else

{Jeśli nie jest cyfrą, sprawdzamy, czy jego kod pasuje do któregoś
 ze znaków wielkich liter.}
if((tekst[licznik_znakow] >= 'A') and
   (tekst[licznik_znakow] <= 'Z')) then
   indeks := ord(tekst[licznik_znakow]) - 54 else
```

```
{Jeśli nie jest wielką literą, sprawdzamy, czy jest małą literą.}
if((tekst[licznik_znakow] >= 'a') and
   (tekst[licznik_znakow] <= 'z')) then
   indeks := ord(tekst[licznik_znakow]) - 60 else

begin
{Jeśli procedura wejdzie do tego bloku kodu, oznaczać to będzie, że
 aktualnie rozpatrywany znak nie jest obsługiwany przez procedurę
 g_wyswietl_tekst; w takim przypadku przechodzimy do rozpatrzenia
 kolejnego znaku.}
x := x + 8;
if (x > 312) then break;
przesuniecie := (y * 320) + x;
continue;
end;

{Maska bitowa służąca do śledzenia ustawionych bitów w bajtach wzorca znaku.}
maska_bitowa := $80;  {80h = 10000000b}

{Malowanie znaku.}
for licznik_bajtow := 0 to 7 do
begin
for licznik_bitow := 0 to 7 do
begin
if ((g_tab_znakow[indeks][licznik_bajtow] and maska_bitowa) =
     maska_bitowa) then
asm
les di, wsk_buforu         {Ładuj adres buforu wideo, segment do ES,
                            offset do DI.}
add di, przesuniecie       {Dodaj przesunięcie do DI.}
mov ah, kolor              {Ładuj kolor do AH.}
mov byte ptr es:[di], ah   {Ładuj AH do komórki pamięci spod adresu
                            ES:DI.}
end;

{Przesuwamy ustawiony bit maski o jedną pozycję w prawo.}
maska_bitowa := maska_bitowa shr 1;

{Przechodzimy do kolejnego piksela malowanej linii.}
przesuniecie := przesuniecie + 1;

end; {koniec pętli for licznik_bitow}

{Cofamy się na początek linii i przeskakujemy o jedną niżej.}
przesuniecie := przesuniecie + 320 - 8;

{Odświeżenie maski bitowej.}
maska_bitowa := $80;  {80h = 10000000b}

end; {koniec pętli for licznik_bajtow}

{Ustawienie przesunięcia na pierwszy piksel kolejnego rysowanego
 znaku; przy okazji sprawdzamy, czy następny znak wyjdzie poza prawą
 krawędź ekranu; jeśli tak, to przerywamy pętlę rysującą znaki.}
x := x + 8;
if (x > 312) then break;  {Przerwanie pętli for licznik_znakow.}
przesuniecie := (y * 320) + x;

end; {koniec pętli for licznik znakow}

end; {koniec procedury g_wyswietl_tekst_13h}
```

Do procedury g_wyswietl_tekst_13h przekazujemy 5 parametrów. Pierwszy zawiera adres buforu wideo, w którym ma zostać narysowany tekst; parametr drugi i trzeci to współrzędne punktu na ekranie, w którym ma zostać wyświetlony pierwszy znak tekstu; parametr czwarty to zmienna typu string zawierająca wypisywany tekst; ostatni parametr określa numer koloru znaków tekstu.

Procedura zawiera nową instrukcję, jest nią continue; jej zadanie polega na natychmiastowym rozpoczęciu wykonywania nowego cyklu pętli, w której się znajduje.

Procedura wykorzystuje również jedną bardzo użyteczną właściwość zmiennych typu string; mimo że rozmiar zmiennych typu string to 256 bajtów, możemy do nich zapisać jedynie 255 znaków, ponieważ pierwszy bajt przeznaczono do określenia ich liczby. Procedura wykorzystuje ten fakt do sprawdzenia, ile znaków zawiera otrzymany przez nią tekst.

Za najważniejszy element procedury uznać możemy pętlę rysującą pojedynczy znak. Sprawdza ona każdy z 64 bitów danego wzorca znaku; jeżeli wykryje ustawiony bit, rysuje w odpowiednim miejscu piksel.

Napiszmy jeszcze jedną procedurę, tym razem bardzo małą; przyda nam się ona w przykładowym programie i w następnych rozdziałach. Jej zadaniem będzie ustawienie pól intensywności jednego wybranego wzorca kolorów. Procedurę nazwiemy g_ustaw_wzorzec_koloru, a oto jej kod:

```
procedure g_ustaw_wzorzec_koloru(nr_wzorca : byte; c, z, n : byte);
begin

{Określamy numer wzorca koloru, który chcemy zmodyfikować.}
port[$3c8] := nr_wzorca;

{Wysyłamy do portu 3c9h intensywność podstawowych kolorów.}
port[$3c9] := c;
port[$3c9] := z;
port[$3c9] := n;

end; {koniec procedury g_ustaw_wzorzec_koloru}
```

Procedury g_wyswietl_tekst_13h oraz g_ustaw_wzorzec_koloru dodajemy do modułu graf_fx. Przykładowy program znajdujący się na listingu 6.3 po uruchomieniu wyświetla na środku ekranu czerwony tekst za pomocą procedury g_wyswietl_tekst_13h, jednak zanim to zrobi, ustala kolor drugiego wzorca koloru na czerwony.

Listing 6.3. *Wyświetlanie tekstu (p_6_3.pas)*

```
program p_6_3;

{// D O Ł Ą C Z O N E   M O D U Ł Y
{////////////////////////////////////////}
uses crt, graf_fx;

{// G Ł Ó W N Y   P R O G R A M
{////////////////////////////////////////}
```

```
begin

{Włączamy tryb graficzny 320×200×256.}
g_ustaw_tryb_vga($13);

{Ustalamy kolor drugiego wzorca koloru palety na czerwony.}
g_ustaw_wzorzec_koloru(2, 63, 0, 0);

{Wyświetlenie tekstu.}
g_wyswietl_tekst_13h(ptr($a000,$0000), 110, 90, 'Rozdzial 6', 2);

{Wyświetlenie tekstu.}
g_wyswietl_tekst_13h(ptr($a000,$0000), 160, 110, 'jjjeeee...', 2);

{Czekanie na naciśnięcie dowolnego klawisza.}
readkey;

end. {koniec programu}
```

Podsumowanie

Przypomnijmy sobie, jakie nowe umiejętności zdołałeś zdobyć w tym rozdziale. Po pierwsze, potrafisz już odciąć komputer od dostępu do klawiatury i samodzielnie przejąć nad nią kontrolę, sprawdzać, które klawisze są naciskane, a które zwalniane. Po drugie, masz władzę nad myszą (król Popiel ma czego pozazdrościć), możesz śledzić jej ruch i stan przycisków. Po trzecie, stworzyliśmy procedurę wypisującą łańcuchy znaków, co ułatwi nam dalszą pracę. Jakość tekstu malowanego przez procedurę `g_wyswietl_tekst` jest dobra, jednak jeżeli zdecydujesz się na stworzenie własnej wymarzonej gry (z pewnością już nad nią pracujesz), zalecam wykorzystanie go tylko do celów roboczych; najlepiej będzie, jeśli stworzysz nową procedurę rysującą litery na podstawie wcześniej specjalnie przygotowanych bitmap; będą dużo ładniej się prezentować.