

5.9 Modyfikacja gry "Kółko i krzyżyk"

Zajmiemy się obecnie grą, której plansza jest widoczna na rys. 5.17 (aplikacja *Do15.bpr*).

Rysunek 5.17: Plansza do gry "suma do 15"

Jej celem jest zaznaczenie cyfr, aby suma trzech wynosiła 15. Kto pierwszy uzyska taki wynik, ten wygrywa. Oczywiście jednym z graczy jest komputer i naszym celem jest napisanie programu, dzięki któremu wygranie z komputerem nie będzie prostym zadaniem.

Przyjrzyjmy się jeszcze cyfrom widocznym na rys. 5.17 (gracz stawia krzyżyki, a komputer kółka). Gracz zaznaczył cyfry 2, 3, 4, 5, a komputer 1, 6, 7, 8. Wygrał komputer, ponieważ suma cyfr 1, 6, 8 wynosi 15. Pamiętajmy, że nie sumujemy wszystkich cyfr, a tylko dowolne trzy spośród zaznaczonych.

Za chwilę pokażemy, że rozpatrywana gra jest równoważna szeroko znanej grze "kółko i krzyżyk", w której jeden z graczy stawia kółka, a

drugi krzyżyki. Wygrywa ten z graczy, który pierwszy ustawi trzy symbole w dowolnym wierszu, kolumnie lub na jednej z przekątnych. Jak wiadomo, bez względu na to kto zaczyna, gra "kółko i krzyżyk" zawsze kończy się remisem, chyba że ktoś popełni błąd. Trzeba jednak bardzo uważać, chwila nieuwagi i porażka nieunikniona. Znacznie trudniejsza jest sytuacja gracza w prezentowanej grze suma do 15". Musi on bowiem w pamięci analizować tablicę, która zawiera niezbędne dane. Z tego też względu zawsze gracz rusza się pierwszy. Ponadto program, który zostanie za chwilę zaprezentowany jest z premedytacją napisany tak, aby dawał szansę wygranej przeciwnikowi. Możliwe jest oczywiście napisanie programu, który grałby w sposób bezbłędny, ale niech to będzie ćwiczenie dla Czytelnika.

Rozważmy teraz następującą tablicę, którą w programie nazwiemy **cyfry**.

```
4 9 2
3 5 7
8 1 6
```

Jest to jedyna tablica o rozmiarach $3 * 3$ mająca własność, że suma trzech liczb leżących w dowolnym wierszu, kolumnie lub jednej z dwóch przekątnych jest równa 15.

Odpowiedniość pomiędzy naszą grą, a grą w kółko i krzyżyk jest teraz oczywista. Wybranie trzech cyfr, których suma jest równa 15, jest równoważne postawieniu na przykład trzech krzyżyków w dowolnym wierszu, kolumnie lub na przekątnej.

Komputer analizuje ruch w tablicy o nazwie **kod**, w której jest zapamiętywane wstawienie kółka lub krzyżyka. Niech kod kółka jest równy 10, a kod krzyżyka to 1. Przypuśćmy teraz, że gracz w pierwszym ruchu wybrał cyfrę 3. Tablica **kod** będzie wtedy wyglądała następująco:

```
0 0 0
1 0 0
0 0 0
```

Odpowiedź komputera to na przykład cyfra 6. A oto tablica **kod**:

```
0 0 0
1 0 0
0 0 10
```

Wyobraźmy sobie teraz, że gracz zaznaczył cyfrę 9, co powoduje wpisanie następujących wartości do tablicy **kod**:

```
0 1 0
1 0 0
0 0 10
```

Komputer wybrał cyfrę 5, co zmienia tablicę kod:

```
0 0 0
1 10 0
0 0 10
```

Gracz, żeby nie przegrać musi teraz zaznaczyć cyfrę 4 (jest konieczne zablokowanie przekątnej lewo w skos). A oto tablica kod:

```
1 1 0
1 10 0
0 0 10
```

Zauważmy, że w tej sytuacji komputer jest już bez szans. Grozi bowiem dostawienie krzyżyka w wierszu o zerowym indeksie (cyfra 2) lub w kolumnie o zerowym indeksie (cyfra 8) i nie ma możliwości zapobieżenia tym dwóm niebezpieczeństwom jednocześnie.

Za chwilę powrócimy jeszcze do strategii, którą zastosujemy w projektowanej aplikacji, a obecnie skoncentrujemy się na zaprojektowaniu odpowiedniej struktury danych. Do wpisywania kółka lub krzyżyka można zastosować różne komponenty, byle tylko było możliwe umieszczenie na nim obrazka. W aplikacji do tego celu zostanie wykorzystany komponent klasy pochodnej od klasy `TImage`. Będzie to klasa pochodna, ponieważ dobrze jest wiedzieć, w którym miejscu w tablicy `cyfry` znajduje się dana cyfra. Na przykład dla cyfry 7 indeks wiersza to 1, a indeks kolumny 2. Oczywiście można za każdym razem przeglądać tablicę `cyfry`, ale jest to rozwiązanie mało eleganckie. W klasie `SpecObraz` pochodnej od klasy `TImage` utworzymy zatem dwa pola o nazwach `i` oraz `j`, gdzie będziemy umieszczać odpowiednie indeksy wiersza i kolumny. A oto deklaracja tej klasy:

```
class SpecObraz: public TImage
{
public:
    int i,j;
    __fastcall SpecObraz(TComponent *AOwner):
        TImage(AOwner) {};
};
```

Deklarację tablicy `kod` oraz `cyfry` należy umieścić w części **private**

deklaracji klasy TForm1. Ponadto powinny tam się znaleźć deklaracje zmiennych `graWToku` i `liczbaRuchow`, sterujących pracą programu oraz deklaracja tablicy obrazki zawierającej wskazania na obiekty klasy `SpecObraz`.

```
private: // User declarations
    int kod[3][3], cyfry[3][3];
    int graWToku, liczbaRuchow;
    SpecObraz * obrazki[9];
```

Możemy teraz skoncentrować się na strategii gry, którą będzie realizował komputer. Zapiszmy ją w punktach:

1. Dwa kółka (w wierszu, kolumnie lub przekątnej) - dostawić trzecie kółko i wygrana.
2. Dwa krzyżyki - dostawić kółko (zapobieżenie wygranej przeciwnika).
3. Jedno kółko i brak krzyżyka - dostawić w sposób przypadkowy drugie kółko.
4. W przeciwnym przypadku wstawić w sposób przypadkowy kółko do całej tablicy.

Strategia ta oczywiście nie zapewnia remisu, bez względu na to jak będzie grał przeciwnik. Realizacja punktów 1 i 2 jest konieczna, natomiast w punkcie 3 należałoby sprawdzić różne możliwości. Specjalnie tego nie robimy, bo co to za gra, w której nie można wygrać. Komputer musi zatem co jakiś czas przegrać.

Zauważmy teraz, że fakt, iż w wierszu, kolumnie i na przekątnej są dokładnie dwa kółka, jest równoważny temu, że suma odpowiednich elementów jest równa 20. Podobnie wystąpienie dokładnie dwóch krzyżyków jest równoważne temu, że suma elementów jest równa 2. Kółko bez żadnego krzyżyka jest wtedy, gdy suma elementów jest równa 10. Warto zatem zaprojektować metodę `Oblicz` typu `bool` z parametrami `s` i `num`, gdzie `s` oznacza sumę elementów, a `num` pozycję, na której trzeba wstawić kółko. Jeżeli metoda `Oblicz` podaje `true`, oznacza to, że istnieje wiersz, kolumna lub przekątna o sumie elementów równej `s`, a w drugim parametrze jest podawana cyfra, którą należy zaznaczyć.

Zaprojektowana niżej metoda `RuchKomp` bazuje na metodzie `Oblicz`. Na przykład konstrukcja:

```
if (Oblicz(20, num)) // dwie dziesiątki
    graWToku = 0;    // zatrzymanie gry
```

oznacza, że znaleziono wiersz, kolumnę lub przekątną z sumą elementów równą 20, czyli są tam umieszczone dwa kółka. Za chwilę dostawimy trzecie, a więc można zatrzymać grę przypisując zmiennej `graWToku` wartość 0. Jeżeli powyższy warunek nie zachodzi, to sprawdzamy następujący:

```
else if (Oblicz(2, num)) ; // dwie dwójki
```

i tak dalej.

A oto treść metody `RuchKomp`:

```
void __fastcall TForm1::RuchKomp()
// ruch komputera
{
    int num; // numer obrazka - wstawienie kółka

    if (Oblicz(20, num)) // dwie dziesiątki
        graWToku = 0;    // zatrzymanie gry
    else if (Oblicz(2, num)) ; // dwie dwójki
    else if (Oblicz(10, num)) ; // jedna dziesiątka
    else
        num = Dowolne(); // dowolny ruch

    // wpisanie kółka do obrazka na pozycji num
    if (num > 0)
        obrazki[num-1]->Picture->LoadFromFile("kolo.bmp");
}
```

Zasygnalizowana już wcześniej metoda `Oblicz` sprawdza, czy istnieje wiersz, kolumna, czy też przekątna o sumie równej `s`. Koniecznie trzeba zwrócić uwagę na sposób wykonywania sprawdzania, który nie polega na braniu pod uwagę najpierw wierszy, potem kolumn, a następnie przekątnych. Działania są celowo przemieszane po to, by zapewnić większą przypadkowość. Można również zastanowić się nad przypadkowym sterowaniem wykonywania poszczególnych działań, ale w tej aplikacji nie zostało to zrealizowane. Nie jest to zresztą konieczne do osiągnięcia zamierzonego efektu.

W metodzie Oblicz wykorzystujemy cztery pomocnicze metody wstawiania elementu do wiersza, kolumny, przekątnej prawo w skos i przekątnej lewo w skos. Treść metody Oblicz jest podana poniżej.

```
bool __fastcall TForm1::Oblicz(int s, int & num)
// sprawdzenie, czy istnieje wiersz, kolumna, czy też
// przekątna o sumie równej s
// jeżeli tak, to podawana jest pozycja num, na której będzie
// wstawione kółko
{
    int i,j,suma;

    // przekątna lewo w skos
    suma = 0;
    for (i=0; i<3; i++)
        suma += kod[2-i][i];

    // jeśli suma wartości jest równa s
    if (suma == s) {
        num = WstawDoLewo();
        return 1;
    }

    // wiersz o indeksie 1
    suma = 0;
    for (j=0; j<3; j++)
        suma += kod[1][j];

    // jeśli suma wartości jest równa s
    if (suma == s) {
        num = WstawDoWiersza(1);
        return 1;
    }

    // kolumna o indeksie 1
    suma = 0;
    for (i=0; i<3; i++)
        suma += kod[i][1];

    // jeśli suma wartości jest równa s
```

```
if (suma == s) {
    num = WstawDoKolumny(1);
    return 1;
}

// wiersz o indeksie 2
suma = 0;
for (j=0; j<3; j++)
    suma += kod[2][j];

// jeśli suma wartości jest równa s
if (suma == s) {
    num = WstawDoWiersza(2);
    return 1;
}

// przekątna prawo w skos
suma = 0;
for (i=0; i<3; i++)
    suma += kod[i][i];

// jeśli suma wartości jest równa s
if (suma == s) {
    num = WstawDoPrawo();
    return 1;
}

// kolumna o indeksie 0
suma = 0;
for (i=0; i<3; i++)
    suma += kod[i][0];

// jeśli suma wartości jest równa s
if (suma == s) {
    num = WstawDoKolumny(0);
    return 1;
}

// wiersz o indeksie 0
```

```
    suma = 0;
    for (j=0; j<3; j++)
        suma += kod[0][j];

    // jeśli suma wartości jest równa s
    if (suma == s) {
        num = WstawDoWiersza(0);
        return 1;
    }

    // kolumna o indeksie 2
    suma = 0;
    for (i=0; i<3; i++)
        suma += kod[i][2];

    // jeśli suma wartości jest równa s
    if (suma == s) {
        num = WstawDoKolumny(2);
        return 1;
    }

    return 0;    // nic nie wstawiono
}
```

Podane niżej cztery metody wykorzystują ten sam algorytm. Polega on na wyznaczeniu liczby zer, w przypadku gdy istnieją dwa zera (trzy nie mogą z założenia). Jest wtedy losowane jedno z nich, a następnie jest wpisywana wartość 10 na miejsce wylosowanego zera. Algorytm jest ten sam, natomiast zmienia się jego realizacja - operujemy na innych indeksach. Ponadto warto zwrócić uwagę na sposób przekazania pozycji, na której ma być wstawione kółko. Otóż jeżeli wartość 10 została wpisana w i-tym wierszu oraz w j-tej kolumnie, to wystarczy wykonać instrukcję:

```
    return cyfry[i][j];
```

aby przekazać odpowiednią wartość.

```
int __fastcall TForm1::WstawDoWiersza(int i)
// wstawienie wartości 10 do i-tego wiersza
{
```

```
int j,k, liczbaZer, wybor;

// wyznaczenie liczby zer
liczbaZer = 0;

for (j=0; j<3; j++)
    if (kod[i][j] == 0)
        liczbaZer++;

// losowanie - tylko gdy są dwa zera
if (liczbaZer == 2)
    wybor = random(199)/100;
else
    wybor = 0;

// wpisanie wartości 10 na miejsce wylosowanego zera
k = 0;
for (j=0; j<3; j++)
    if (kod[i][j] == 0) {
        if (k == wybor) {
            kod[i][j] = 10;
            return cyfry[i][j];
        }
        k++;
    }

return -1;
}

int __fastcall TForm1::WstawDoKolumny(int j)
// wstawienie wartości 10 do j-tej kolumny
{
    int i, k, liczbaZer, wybor;

    // wyznaczenie liczby zer
    liczbaZer = 0;

    for (i=0; i<3; i++)
        if (kod[i][j] == 0)
```

```
        liczbaZer++;

// losowanie - tylko gdy są dwa zera
if (liczbaZer == 2)
    wybor = random(199)/100;
else
    wybor = 0;

// wpisanie wartości 10 na miejsce wylosowanego zera
k = 0;
for (i=0; i<3; i++)
    if (kod[i][j] == 0) {
        if (k == wybor) {
            kod[i][j] = 10;
            return cyfry[i][j];
        }
        k++;
    }
return -1;
}

int __fastcall TForm1::WstawDoPrawo()
// wstawienie wartości 10 do przekątnej prawo w skos
{
    int i, k, liczbaZer, wybor;

// wyznaczenie liczby zer
liczbaZer = 0;

for (i=0; i<3; i++)
    if (kod[i][i] == 0)
        liczbaZer++;

// losowanie - tylko gdy są dwa zera
if (liczbaZer == 2)
    wybor = random(199)/100;
else
    wybor = 0;
```

```
// wpisanie wartości 10 na miejsce wylosowanego zera
k = 0;
for (i=0; i<3; i++)
    if (kod[i][i] == 0) {
        if (k == wybor) {
            kod[i][i] = 10;
            return cyfry[i][i];
        }
        k++;
    }

return -1;
}

int __fastcall TForm1::WstawDoLewo()
// wstawienie wartości 10 do przekątnej lewo w skos
{
    int i, k, liczbaZer, wybor;

    // wyznaczenie liczby zer
    liczbaZer = 0;

    for (i=0; i<3; i++)
        if (kod[2-i][i] == 0)
            liczbaZer++;

    // wpisanie wartości 10 na miejsce wylosowanego zera
    if (liczbaZer == 2)
        wybor = random(199)/100;
    else
        wybor = 0;

    // wpisanie wartości 10 na miejsce wylosowanego zera
    k = 0;
    for (i=0; i<3; i++)
        if (kod[2-i][i] == 0) {
            if (k == wybor) {
                kod[2-i][i] = 10;
                return cyfry[2-i][i];
            }
        }
}
```

```
        }
        k++;
    }
    return -1;
}
```

Metoda Dowolne jest stosowana wtedy, gdy trzeba wpisać wartość 10 na miejsce któregoś zera w całej tablicy i to w dodatku w sposób przypadkowy. Przypadkowość zapewnia, że komputer na takie same ruchy gracza będzie reagował za każdym razem inaczej.

```
int __fastcall TForm1::Dowolne()
// wstawienie wartości 10 na miejsce wylosowanego
// zera w całej tablicy
{
    int i,j,k,liczbaZer,wybor;

    // wyznaczenie liczby zer
    liczbaZer = 0;

    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        if (kod[i][j] == 0)
            liczbaZer++;

    // wylosowanie zera (które z kolei)
    wybor = random(liczbaZer);

    // wpisanie wartości 10 na miejsce wylosowanego zera
    k=0;
    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        if (kod[i][j] == 0) {
            if (k == wybor) {
                kod[i][j] = 10;
                return cyfry[i][j];
            }
            k++;
        }
}
```

```
    return -1;
}
```

W metodzie `FormCreate` nadamy wartości początkowe tablicom `kod` oraz `cyfry` i przede wszystkim utworzymy 9 obiektów klasy `SpecObraz` (klasa pochodna od klasy `TImage`). Niezbędne własności ustawiamy w sposób programowy. Dla wszystkich obiektów metoda wykonywana po zajściu zdarzenia `OnClick` to `Gracz`. Warto zwrócić uwagę na sposób ustawienia własności `i, j` zdefiniowanych w klasie `SpecObraz`. Trzeba przejrzeć całą tablicę `cyfry`, aby znaleźć odpowiednie indeksy. Działanie to wykonuje się jednak tylko raz. A oto odpowiedni fragment programu (obiekt o indeksie `k` jest przyporządkowany cyfrze `k+1`):

```
    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        if (cyfry[i][j] == k+1) {
            obrazki[k]->i = i;
            obrazki[k]->j = j;
        }
```

Cyfry od 1 do 9 są wyświetlane w komponentach klasy `TLabel`. W metodzie obiekty te są ustawiane na formularzu poprzez określenie własności `Top` oraz `Left` by uniknąć wykonywania tych czynności podczas projektowania formularza. Metoda `FormCreate` jest przedstawiona poniżej.

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int i,j,k;

    // inicjalizacja generatora liczb losowych
    randomize();

    graWToku = 1;    // gra w toku
    liczbaRuchow = 0;

    Komunikat->Visible = false;

    // w tablicy same zera
    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
```

```
    kod[i][j] = 0;

// wypełnienie tablicy cyfry
cyfry[0][0] = 4;
cyfry[0][1] = 9;
cyfry[0][2] = 2;

cyfry[1][0] = 3;
cyfry[1][1] = 5;
cyfry[1][2] = 7;

cyfry[2][0] = 8;
cyfry[2][1] = 1;
cyfry[2][2] = 6;

// utworzenie 9 obrazków
for (k=0; k<9; k++) {
    obrazki[k] = new SpecObraz(this);
    obrazki[k]->Parent = this;

    obrazki[k]->Left = 50 + k*60;
    obrazki[k]->Top = 200;

    obrazki[k]->Width = 30;
    obrazki[k]->Height = 30;

    obrazki[k]->Visible = true;
    obrazki[k]->Stretch = true;

    // załadowanie obrazka
    obrazki[k]->Picture->LoadFromFile("puste.bmp");

    // podanie nazwy metody wykonywanej po zajściu
    // zdarzenia OnClick
    obrazki[k]->OnClick = Gracz;

    // nadanie wartości własnościom i,j
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
```

```
        if (cyfry[i][j] == k+1) {
            obrazki[k]->i = i;
            obrazki[k]->j = j;
        }
    }

    // wyświetlenie cyfr w odpowiednim miejscu na formularzu
    Label1->Left = 60;
    Label1->Top = 165;

    Label2->Left = 120;
    Label2->Top = 165;

    Label3->Left = 180;
    Label3->Top = 165;

    Label4->Left = 240;
    Label4->Top = 165;

    Label5->Left = 295;
    Label5->Top = 165;

    Label6->Left = 355;
    Label6->Top = 165;

    Label7->Left = 415;
    Label7->Top = 165;

    Label8->Left = 475;
    Label8->Top = 165;

    Label9->Left = 535;
    Label9->Top = 165;
}
```

Metoda Gracz jest wykonywana po zajściu zdarzenia OnClick dla każdego obiektu klasy SpecObraz. Po wyznaczeniu numeru obiektu, na którym naciśnięto przycisk myszki, należy sprawdzić, czy przypadkiem w obiekcie tym nie jest już umieszczone kółko lub krzyżyk. W tym celu wystarczy sprawdzić w tablicy kodów, jaka wartość stoi na odpowiedniej

pozycji. A odpowiednią pozycję można bardzo łatwo wyznaczyć, ponieważ każdy obiekt klasy SpecObraz ma dwie własności podające numer wiersza i kolumny. Wystarczy zatem napisać:

```
if (kod[obrazki[nr]->i][obrazki[nr]->j] == 10 ||
    kod[obrazki[nr]->i][obrazki[nr]->j] ==1)
    return;
```

i to wszystko.

Następnie do tablicy kodów na odpowiedniej pozycji należy wpisać wartość 1. I znowu dokonuje się tego w bardzo prosty sposób:

```
kod[obrazki[nr]->i][obrazki[nr]->j] = 1;
```

Kolejny istotny problem to sprawdzenie, czy gracz wstawił trzy krzyżyki do dowolnego wiersza, kolumny lub przekątnych (w tablicy kod). Problem ten można rozwiązać wykorzystując metodę Oblicz. Jeżeli bowiem wywołamy tą metodę z następującymi parametrami:

```
Oblicz(3,pom)
```

to wartość **true** zostanie zwrócona tylko wtedy, gdy w którymś wierszu, kolumnie lub przekątnej są trzy jedyńki.

Jeżeli gracz nie wygrywa, to trzeba sprawdzić, czy jest remis. I właśnie do tego celu służy zmienna `liczbaRuchow`. Jeśli gracz wykonał 5 ruchów i nie nastąpiło rozstrzygnięcie, to znaczy, że jest remis.

I ostatnie już wyjaśnienie. Komputer wygrywa, jeżeli po wykonaniu przez niego ruchu gra jest zatrzymana, o czym świadczy wartość zmiennej `graWToku` równą 0.

```
void __fastcall TForm1::Gracz(TObject *Sender)
// metoda wykonywana po naciśnięciu przycisku myszki na
// obiektach klasy SpecObraz
{
    int i,j, nr, pom;
    nr = 0;

    // jeśli koniec gry
    if (!graWToku) return;

    // wyznaczenie numeru obiektu, na którym naciśnięto
```

```
// przycisk myszki
for (i=0; i<9; i++)
    if (obrazki[i] == Sender)
        nr = i;

// jeśli w danym obiekcie jest już kółko lub krzyżyk
if (kod[obrazki[nr]->i][obrazki[nr]->j] == 10 ||
kod[obrazki[nr]->i][obrazki[nr]->j] ==1)
    return;

// obliczenie liczby wykonanych ruchów przez gracza
liczbaRuchow++;

// wpisanie krzyżyka
obrazki[nr]->Picture->LoadFromFile("krzyz.bmp");

// do tablicy kodów wpisujemy 1
kod[obrazki[nr]->i][obrazki[nr]->j] = 1;

if (Oblicz(3,pom)) {
    // są trzy jedynki
    Komunikat->Visible = true;
    Komunikat->Caption = "GRATULACJE - WYGRAŁEŚ";
    graWToku = 0; // koniec gry
} else {
    if (liczbaRuchow == 5) {
        // wykonano wszystkie ruchy bez rozstrzygnięcia
        Komunikat->Visible = true;
        Komunikat->Caption = "REMIS";
    }

    RuchKomp();

    if (!graWToku) {
        // komputer zatrzymał grę
        Komunikat->Visible = true;
        Komunikat->Caption = "NIESTETY PRZEGRAŁEŚ";
    }
}
```

```
}
```

Ostatnia już prosta metoda to `NowaClick` obsługująca rozpoczęcie nowej gry.

```
void __fastcall TForm1::NowaClick(TObject *Sender)
// obsługa rozpoczęcia nowej gry
{
    int i,j,k;

    graWToku = 1;
    liczbaRuchow = 0;

    Komunikat->Visible = false;

    // wyzerowanie tablicy kodów
    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
        kod[i][j] = 0;

    // załadowanie pustych obrazków
    for (k=0; k<9; k++)
        obrazki[k]->Picture->LoadFromFile("puste.bmp");
}
```

Nie możemy jeszcze zapomnieć o dodaniu do części **private** deklaracji klasy `TForm1` poniższych prototypów zaprojektowanych metod.

```
void __fastcall RuchKomp();
bool __fastcall Oblicz(int, int &);
int __fastcall WstawDoWiersza(int);
int __fastcall WstawDoKolumny(int);
int __fastcall WstawDoPrawo();
int __fastcall WstawDoLewo();
int __fastcall Dowolne();
```

Ponadto w części **published** powinna się znaleźć deklaracja metody `Gracz`. Jest to metoda wykonywana po zajściu zdarzenia `OnClick` i dlatego jej deklaracja nie może być umieszczona w części **private**.

```
__published: // IDE-managed Components  
...  
void __fastcall Gracz(TObject *Sender);
```