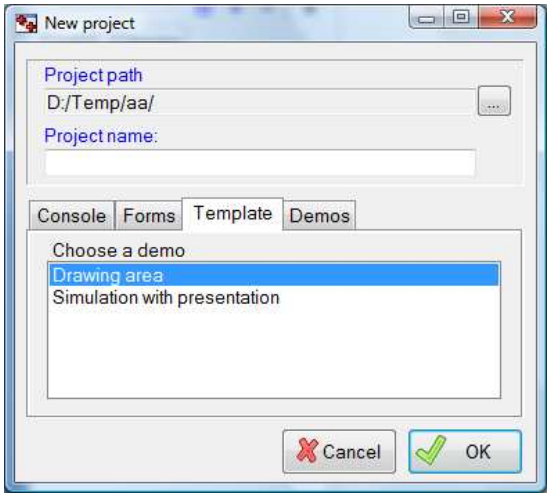
Szybkie tworzenie grafiki w GcIde

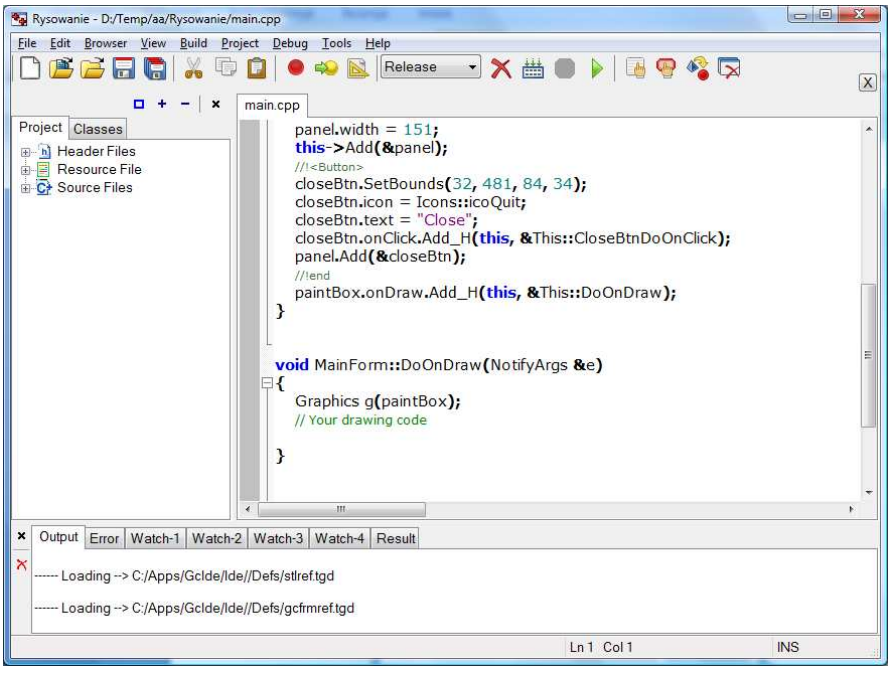
Opracował: Ryszard Olchawa

Poniższy opis dotyczy aplikacji okienkowej w systemie Windows lub Linux bazującej na obiektowej bibliotece „rofrm" stworzonej w środowisku GcIde. Kod źródłowy takiej aplikacji może być kompilowany i uruchamiany bez jakiejkolwiek modyfikacji w obu systemach (Windows i Linux), niezależnie od tego w którym był utworzony. Do uruchomienia skompilowanej aplikacji na danej platformie nie są wymagane żadne dodatkowe biblioteki dynamiczne.

Projekt zawierający prostą grafikę 2D w środowisku GcIde najprościej utworzyć w oparciu o zdefiniowany szablon: „Drawing area". Wybierając polecenie „New Project" z menu „File" uruchomimy prosty kreator projektów:



Szablon: „Drawing area" znajdziemy na zakładce „Template". W polu „Project path" należy ustawić katalog docelowy w którym zostanie zapisany projekt, natomiast w polu „Project name:" wpisać nazwę tworzonego projektu np.: „Rysowanie".
Uwaga! Nazwa projektu nie powinna zawierać spacji ani znaków specjalnych za wyjątkiem znaku podkreślenia: _ .Po kliknięciu przycisku „OK." zostanie wygenerowany szkielet naszej aplikacji:

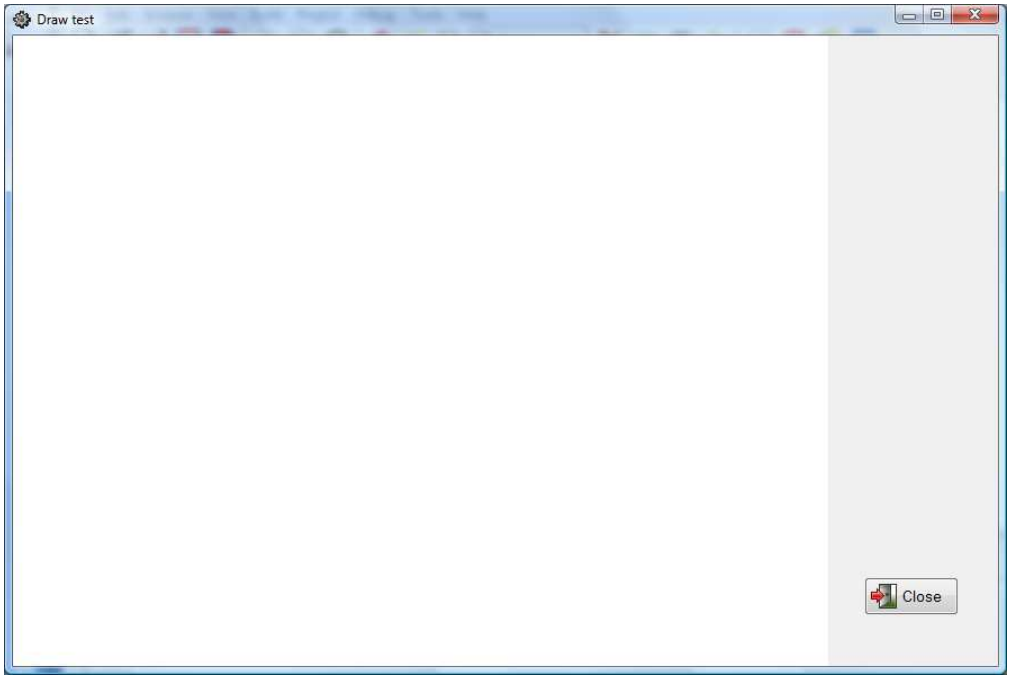


Projekt kompilujemy klikając na przycisk „Make”:

dostępnym na pasku narzędziowym. W czasie trwania kompilacji nieaktywny przycisk po prawej staje się aktywny (umożliwia awaryjne przerwanie kompilacji)

Po zakończeniu kompilacji przycisk ten staje się ponownie nieaktywny (szary). Skompilowaną aplikację możemy uruchomić przyciskiem „Run”

W przypadku naszej aplikacji okno główne wygląda tak:



Biały obszar po lewej stronie to „paintBox”- obiekt na którym będziemy tworzyć grafikę. Prawa strona zawiera obiekt „panel” z przyciskiem zamykającym aplikację. Po zamknięciu aplikacji wracamy do środowiska GcIde zawierającego kod źródłowy naszej aplikacji. Metoda **MainForm::DoOnDraw(NotifyArgs &e)** odpowiada za tworzenie grafiki w obszarze painBox’a. Aktualnie jej postać jest następująca:

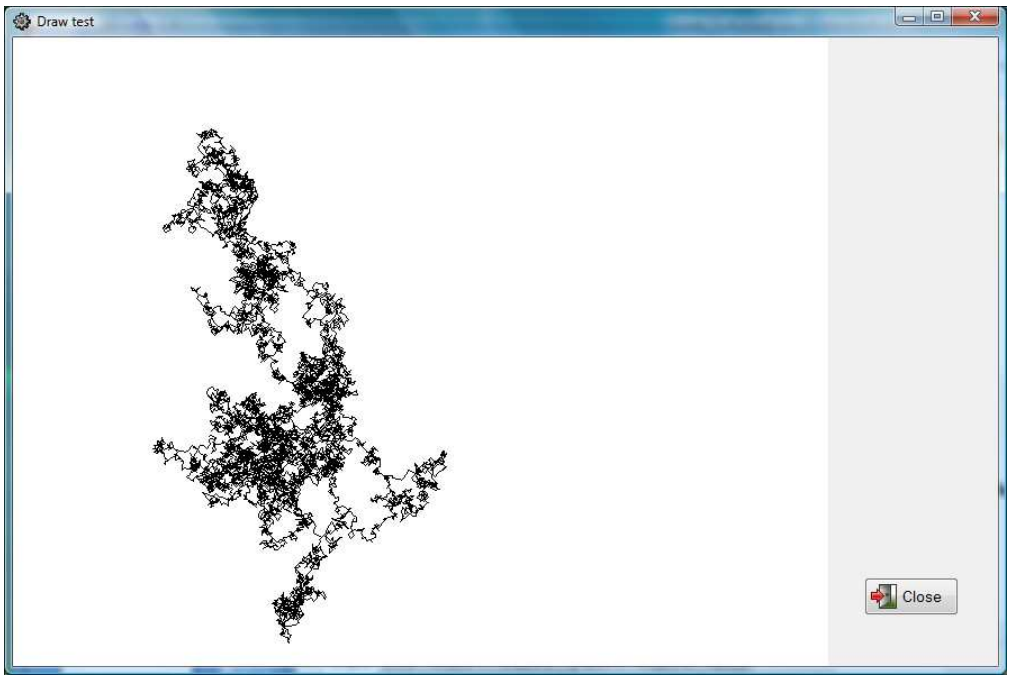
```
void MainForm::DoOnDraw(NotifyArgs &e)
{
   Graphics g(paintBox);
   // Your drawing code

}
```

W miejscu oznaczonym komentarzem wpisujemy nasz kod tworzący obraz np.

```
void MainForm::DoOnDraw(NotifyArgs &e)
{
   Graphics g(paintBox);
   // Your drawing code
   double x =300, y=300;
   double krok = 8;
   g.MoveTo(x, y);
   for(int i=0; i<10000; ++i) {
      x += krok*Random()- krok/2;
      y += krok*Random()- krok/2;
      g.LineTo(x,y);
   }
}
```

Metody „MoveTo” i „LineTo” jak i pozostałe służące do tworzenia grafiki zostaną opisane w dalszej części opracowania. Funkcja „Random” generuje liczbę pseudolosową z przedziału 0..1.

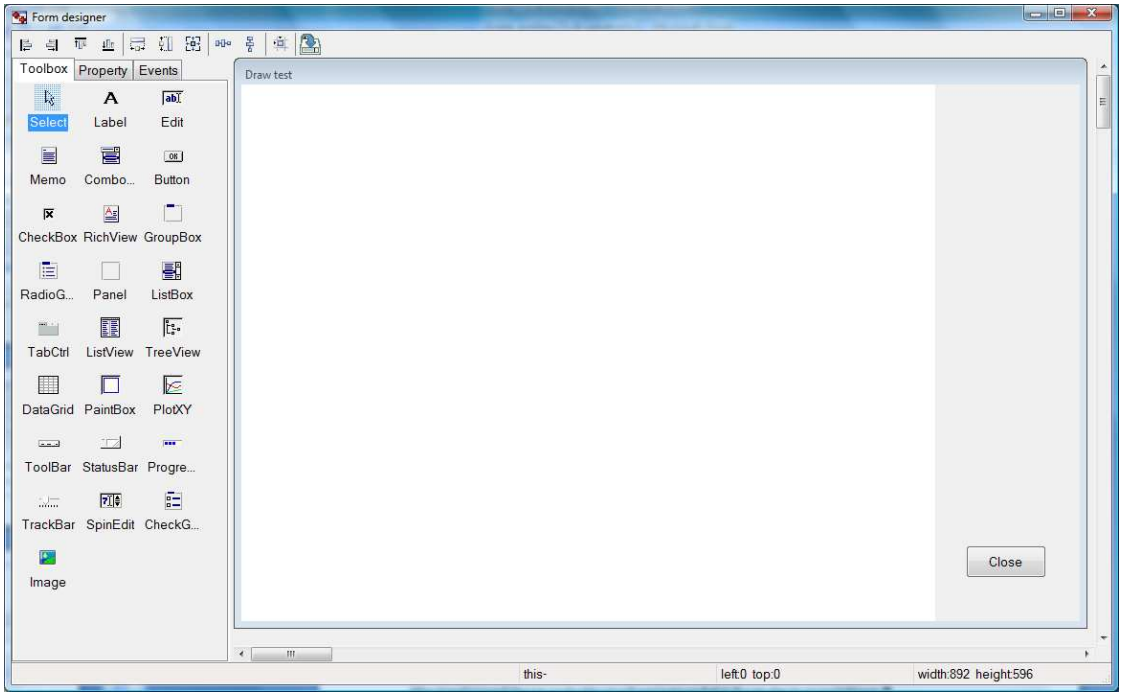
Po ponownym skompilowaniu i uruchomieniu otrzymamy coś w rodzaju:



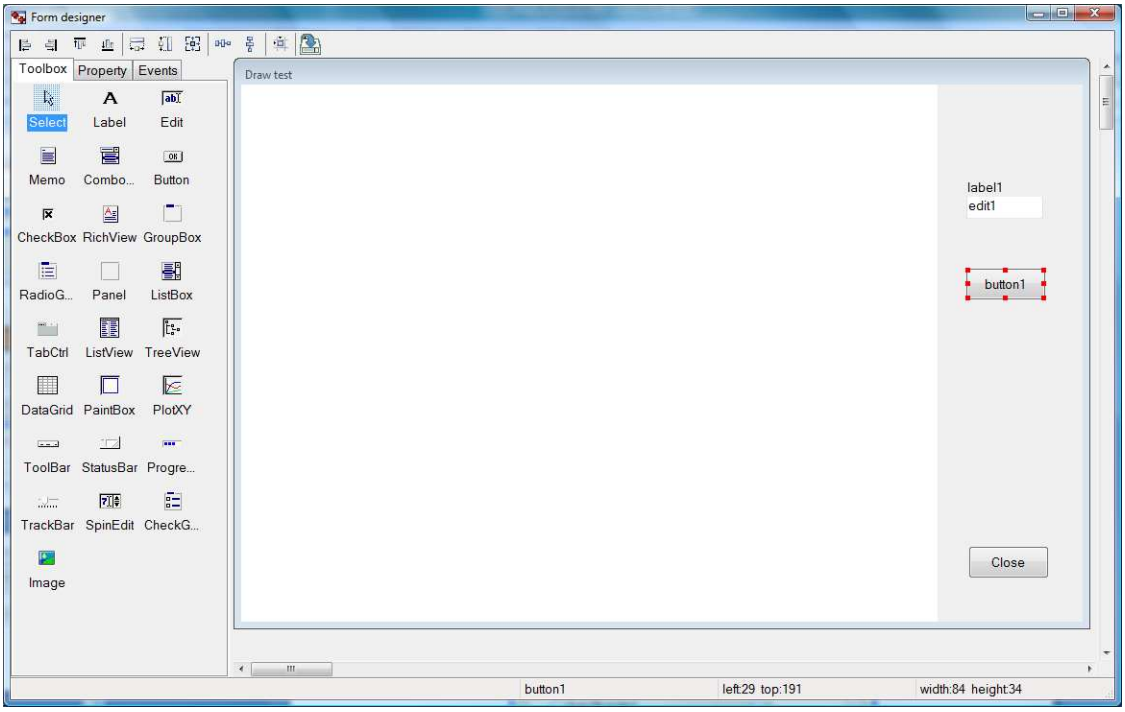
Często zachodzi potrzeba wprowadzenia parametru który wpływa na tworzony obraz, np. chcielibyśmy mieć możliwość zmiany parametru „krok” i wygenerowania nowego obrazu.

Aby zrealizować nasze zadanie uruchamiamy edytor formularzy przyciskiem:

Otworzy się okno edytora:



Klikając na odpowiednią kontrolkę dostępną na palecie po lewej stronie edytora, a następnie na panel na naszym formularzu, umieszczamy kolejno kontrolki „Label”, „Edit”, „Button” jak poniżej:



Właściwości naszych kontrolek możemy zmienić za pomocą edytora właściwości – zakładka „Property.”

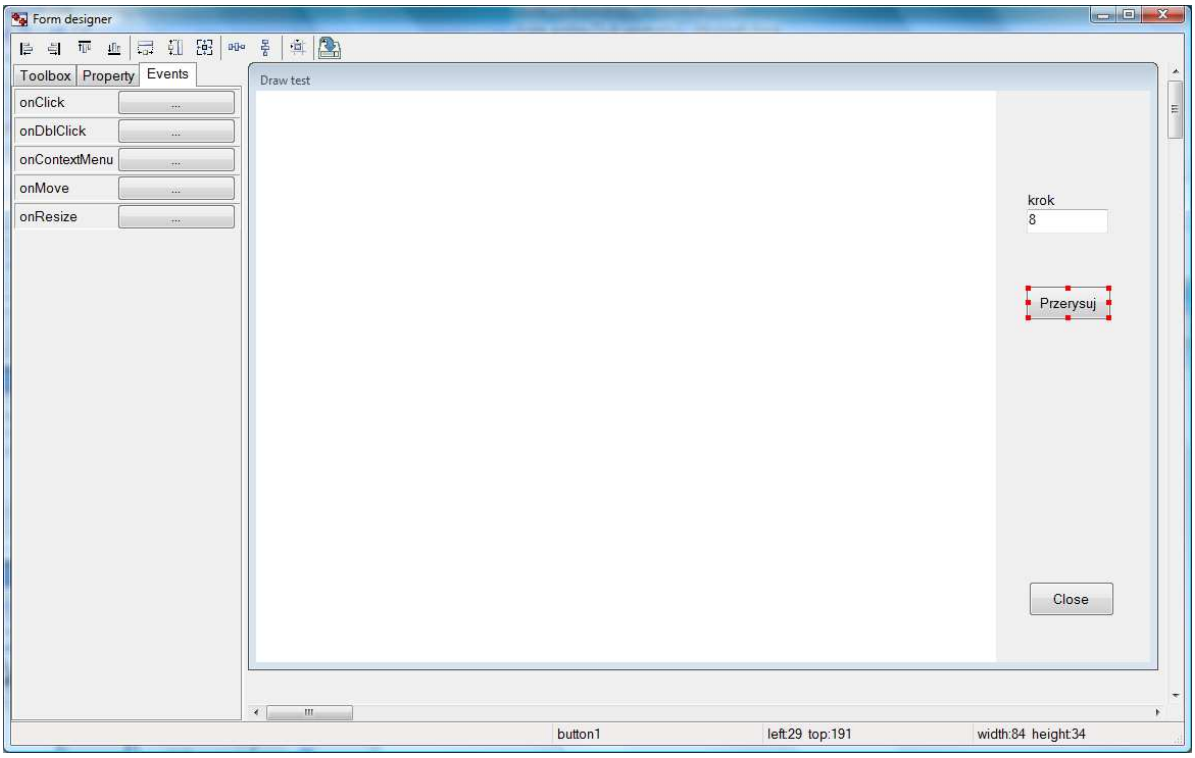
Zmieniamy kolejno:

label1: „text” na „krok”

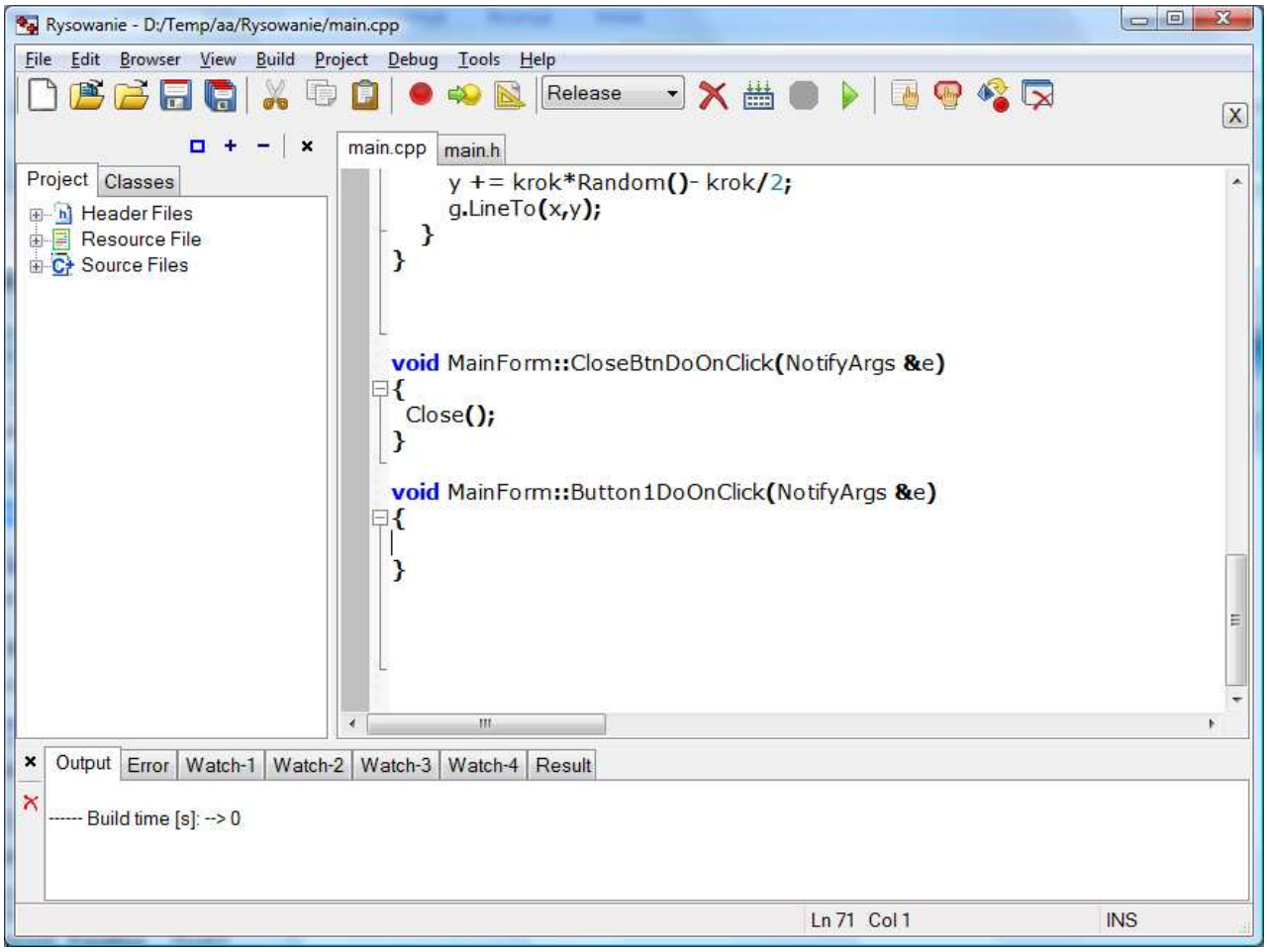
edit1: „name” na „krokEd” oraz „text” na „8”

button1: „text” na „Przerysuj”

Po dokonaniu tych ustawień przechodzimy na zakładkę „Events” i po zaznaczeniu przycisku z napisem „Przerysuj”, klikamy na przycisk „onClick” na zakładce „Events”, w celu wygenerowania procedury obsługi zdarzenia „Click” zaznaczonego przycisku:



Po kliknięciu na przycisk „onClick” edytor formularzy zostanie automatycznie zamknięty i wygenerowany prototyp metody - wywoływanej przez system, w momencie kliknięcia na przycisk „Przerysuj”. Metoda ta zostanie dodana na końcu pliku „main.cpp”



W treści tej metody, wywołamy metodę obiektu „painBox” wymuszającą ponowne rysowanie:

```
void MainForm::Button1DoOnClick(NotifyArgs &e)
{
   paintBox.Redraw();
}
```

Aby umożliwiać zmianę parametru modyfikujemy kod metody:

```
void MainForm::DoOnDraw(NotifyArgs &e)
{
   Graphics g(paintBox);
   // Your drawing code
   g.Clear();
   double x =300, y=300;
   double krok = ToFloat(krokEd.text);
   g.MoveTo(x, y);
   for(int i=0; i<10000; ++i) {
      x += krok*Random()- krok/2;
      y += krok*Random()- krok/2;
      g.LineTo(x,y);
   }
}
```

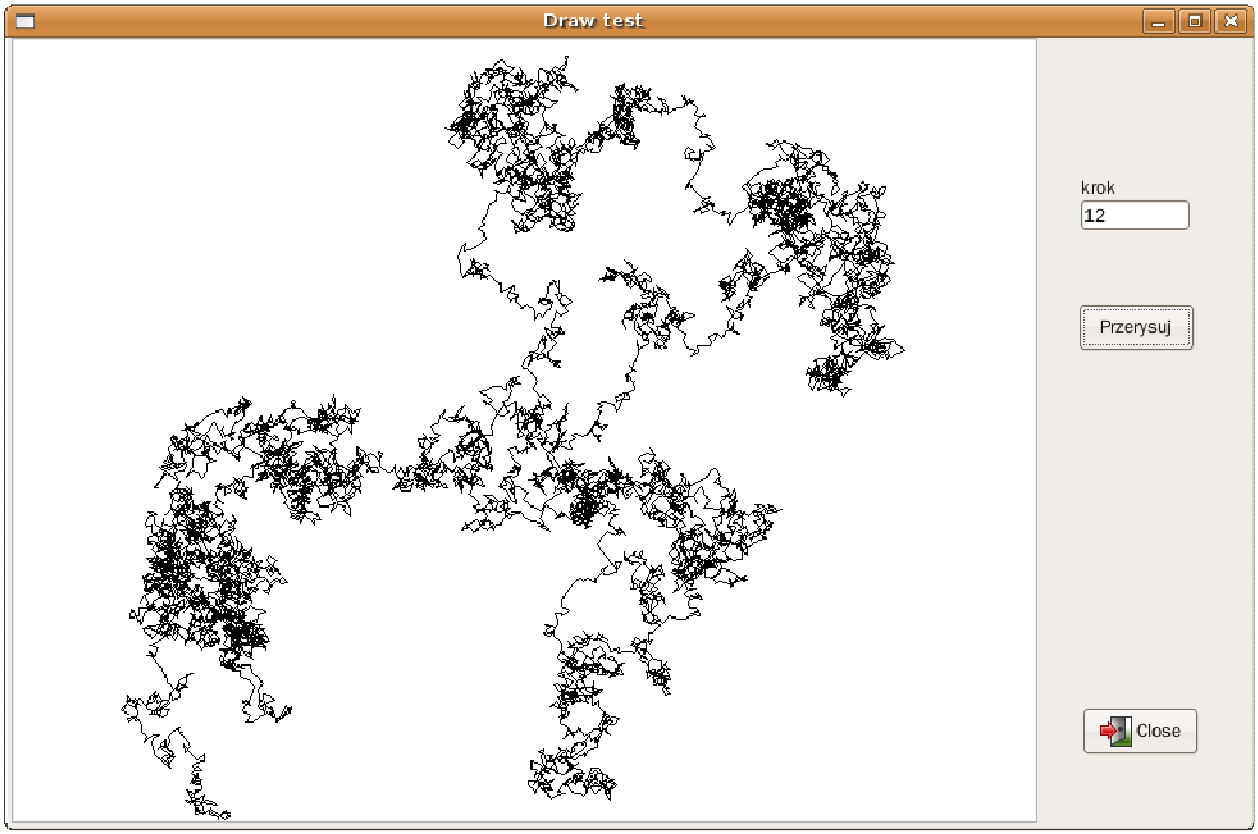
gdzie wprowadzono dwie zmiany:
(1) wywołano metodę „Clear" która czyści poprzednią zawartość paintBox'a (bez niej kolejne obrazy będą nakładały się na siebie),
(2) wartość zmiennej „krok" nadawana jest na podstawie tekstu wpisanego w okienku „krokEd" (funkcja „ToFloat" przekształca zmienną tekstową do zmiennej typu double).

Jeżeli kod odpowiedzialny za rysowanie chcemy przenieść do wydzielonej procedury(funkcji), musimy do tej procedury(funkcji) przekazać referencję do obiektu „Graphics" np.:

```
void MojeRysuj(Graphics &g, double krok)
{
   g.Clear();
   double x =300, y=300;
   g.MoveTo(x, y);
   for(int i=0; i<10000; ++i) {
      x += krok*Random()- krok/2;
      y += krok*Random()- krok/2;
      g.LineTo(x,y);
   }
}

void MainForm::DoOnDraw(NotifyArgs &e)
{
   Graphics g(paintBox);
   // Your drawing code
   MojeRysuj(g, ToFloat(krokEd.text));
}
```

Po przeniesieniu plików źródłowych naszego projektu (katalog „Rysowanie" a w nim pliki: „Rysowanie.gpr", „main.cpp", main.h", main.rc") do systemu Linux i ponownym przekompilowaniu
Uwaga!!! Nie wolno przenosić zawartości podkatalogu „out-rel" ani „out-dbg"

zawierających kompilat projektu dla Windows.
nasza aplikacja wygląda tak:



Domyślnie kolor linii jest czarny aby zmienić kolor na inny należy zdefiniować obiekt lub kilka obiektów Pen a następnie uaktywniać je metodą „SetPen” np.:

```
void MainForm::DoOnDraw(NotifyArgs &e)
{
  Graphics g(paintBox);
  // Your drawing code
  g.MoveTo(100,100);
  Pen p1(Color(255, 0, 0)), p2(Color(0,100, 100), 3);
  g.SetPen(p1);
  g.LineTo(300,100);
  g.SetPen(p2);
  g.LineTo(300,300);
  g.SetPen(p1);
  g.LineTo(100,300);
}
```

W powyższym przykładzie Color zadawany jest poprzez podanie składowych RGB (red, Green, blue), wartość każdej z nich musi mieścić się w przedziale 0..255. Dodatkowy opcjonalny parametr w „p2” określa grubość lini (w tym przypadku 3 piksele).

W podobny sposób określa się kolor wypełniania obszarów tworząc pędzle obiekty „Brush” i uaktywniając je metodą „SetBrush”

Więcej przykładów można znaleźć w aplikacjach demonstracyjnych dołączonych do środowiska „GcIde”

Opis metod obiektu Graphics

void SetPen(const Pen& pen)

Ustaw nowe pióro

Ustawienie nowego pióra (%Pen) umożliwia zmianę koloru, grubości i stylu rysowanej linii.

void SetBrush(const Brush& brush)

Ustaw nowy pędzel

Ustawienie nowego pędzla umożliwia zmianę koloru i stylu wypełnianego obszaru.

void SetFont(const Font& font)

Ustaw nową czcionkę

Ustawienie nowej czcionki umożliwia zmianę koloru, wielkości i stylu rysowanego tekstu.

void MoveTo(int x, int y)

Ustaw nową pozycję kursora graficznego.

Ustawienie nowej pozycji kursora nie powoduje powstania jakiegokolwiek śladu na ekranie.

void LineTo(int x, int y)

Rysuj linię od bieżącej pozycji kursora do punktu [x,y]

Pozycja [x,y] staje się automatycznie nową pozycją kursora.

void Rectangle(int x1, int y1, int x2, int y2)

Rysuje prostokąt.

Rysowany prostokąt wyznaczają przeciwległe wierzchołki współrzędne [x1,y1] oznaczają prawy-góry wierzchołek a [x2,y2] lewy dolny Pozycja .kursora nie ulega zmianie

void Polyline(const Point p[], int count)

Rysuje łamaną łączącą punkty podane w tablicy p[]

Parametr count musi podawać liczbę punktów podanych w tablicy. Pozycja kursora nie ulega zmianie

void Polygon(const Point p[], int count)

Rysuje i wypełnia obszar wielokąta.

Punkty podane w tablicy p[] definiują wierzchołki wypełnianego wielokąta. Do wypełniania używany jest aktualny pędzel (Brash)

void FillRect(int x1, int y1, int x2, int y2)

Rysuje i wypełnia obszar prostokąta.

Współrzędne [x1,y1] oznaczają prawy-góry wierzchołek a [x2,y2] lewy-dolny wypełnianego prostokąta. Do wypełniania używany jest aktualny pędzel (Brash)

void RoundRect(int x1, int y1, int x2, int y2, int wd, int ht)

Rysuje i wypełnia prostokąt o zaokrąglonych wierzchołkach.

Współrzędne [x1,y1] oznaczają prawy-góry wierzchołek a [x2,y2] lewy-dolny wypełnianego prostokąta. Wielkość zaokrąglenia określają wd i ht .Obszar ten jest wypełniany, aktualnym pędzlem (Brash)

void Ellipse(int x1, int y1, int x2, int y2)

Rysuje i wypełnia elipsę.

Współrzędne [x1,y1] oznaczają prawy-góry wierzchołek a [x2,y2] lewy-dolny prostokąta ograniczającego (opisanego na elipsie). Obszar elipsy .wypełniany jest aktualnym pędzlem (Brash)

void Circle(int x, int y, int r)

Rysuje i wypełnia okrąg.

Współrzędne [x,y] określają środek okręgu a r promień.

void Arrow(int x1, int y1, int x2, int y2)

Rysuje linię zakończoną grotem

Linia rysowana jest od punktu [x1,y1] do punktu [x2,y2] i kończy się grotem.

void SetPixel(int x, int y, Color col)

Ustaw kolor piksela w punkcie x,y

Parametr col określa kolor tego pixela.

Color GetPixel(int x, int y)

Funkcja która zwraca kolor piksela w [x,y]

void FillSolidRect(int x, int y, int cx, int cy, const Color clr)

Rysuje i wypełnia obszar prostokąta zadanym kolorem.

Dwie przeciążone metody do szybkiego wypełniania obszaru. Do wypełniania nie jest używany aktualny pędzel, lecz pędzel o kolorze clr.

void FillSolidRect(Range &rect, const Color clr)

Rysuje i wypełnia obszar prostokąta zadanym kolorem.

Metoda przeciążona używająca parametru Range

```
void Draw(int left, int top, int width, int height, Bitmap &src)
```

Kopiuje obraz mapy bitowej

```
void Clear(Color color=Color::cWhite)
```

Wypełnia cały dostępny obszar zadanym kolorem

Domyślnie kolor tła jest biały.