Тема 4.5 Мультимедийные возможности системы программирования

Высокоуровневые методы библиотеки .NET едва ли подойдут для написания насыщенных быстрым движением и графикой мультимедийных или игровых приложений - для этого есть DirectX и OpenGL. Тем не менее, иногда бывает нужно куда-то передвинуть или перетащить картинку, "оживить" приложение небольшой покадровой анимацией и т.п. В качестве примера выполним несколько типовых действий по реализации простой анимации в .NET на C++/CLI.

Общий подход к анимации следующий: на форму добавляется или программно создаётся компонента Timer. Его единственное событие Tick выполняется через заданный в миллисекундах интервал времени Interval. Обработчик события Tick отслеживает, нужна ли перерисовка каких-либо объектов на форме и, при необходимости, вызывает соответствующие методы. Самый простой способ - выполнить метод Invalidate() формы, отправляющий сообщение методу Paint, отвечающему за перерисовку.

1. Картинка следует за курсором мыши

Создав решение Windows Forms на C++, поместим в папку ИмяРешения/Debug подходящую картинку (в коде предполагается имя файла butterfly.png, фон картинки прозрачный, прикреплена внизу страницы).

Опишем в классе формы необходимые объекты:

```
Image ^img; //картинка
Point imgPoint; //точка вывода картинки
Point mousePoint; //цель движения, зависящая от координат мыши
int mouseMode; //переключатель режима "движение картинки за мышью"
```

Дадим им начальные значения в конструкторе формы, после оператора InitializeComponent():

```
this->DoubleBuffered = true;
img = Bitmap::FromFile(Application::StartupPath + "\\butterfly.png");
imgPoint =  Point(0, 0);
mousePoint =  Point();
mouseMode = 1;
```

Обратите внимание, что для формы включена *двойная буферизация* графики - это позволит избежать мерцания картинки при её движении. При относительно медленном процессе отрисовки объектов непосредственно на канве, где они отображаются, часто возникает эффект мерцания изображения. Избежать его позволяет двойная буферизация, идея которой состоит в следующем: объект рисуется на невидимой канве, а затем законченный объект быстрым копированием из одной области оперативной памяти в другую помещается на канву отображения. В библиотеку .NET двойная буферизация уже встроена, для её использования достаточно установить в значение true свойство DoubleBuffered формы.

По событию Load формы программно создадим и инициализируем таймер, который будет управлять процессом движения картинки. Обратите внимание, что

обработчик единственного имеющегося у таймера события Tick, происходящего с частотой в Interval миллисекунд, тоже назначен программно.

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
 Timer ^ timer1 = gcnew Timer();
 timer1->Interval = 5;
 timer1->Enabled = true;
 timer1->Tick += gcnew EventHandler(this, &Form1::timer1_Tick);
}
```

А вот и сам этот метод, добавленный в класс формы:

```
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs ^e) {
 if (mouseMode) {
  MoveImg(mousePoint);
  this->Invalidate(); //самое главное... вызовет, в том числе, обработчик Paint
 }
}
```

Метод MoveImg мы также напишем сами, он будет просто менять координаты, предназначенные для левого верхнего угла картинки:

```
private: void MoveImg(Point point) {
 if (imgPoint.X > point.X) imgPoint.X --;
 if (imgPoint.X < point.X) imgPoint.X ++;
 if (imgPoint.Y > point.Y) imgPoint.Y --;
 if (imgPoint.Y < point.Y) imgPoint.Y ++;
}
```

Кто будет непосредственно заниматься отрисовкой? Конечно, обработчик события Paint, который мы создадим из стандартного окна Свойства:

```
private: System::Void Form1_Paint(System::Object^  sender,
                                  System::Windows::Forms::PaintEventArgs^  e) {
 e->Graphics->DrawImage(img, imgPoint);
}
```

По событию MouseDown (нажатию кнопки мыши) будем переключать режим движения картинки, то есть, кликнем раз - движение за курсором прекратится, а следующий клик возобновит движение и т.д.

```
private: System::Void Form1_MouseDown(System::Object^  sender,
 System::Windows::Forms::MouseEventArgs^  e) {
 mouseMode ^=1;
}
```

Наконец, обработчик события перемещения мыши будет следить за тем, куда нужно смещаться картинке. Обратите внимание на замечание о том, какой должна быть цель движения, чтобы картинка останавливалась "прямо под курсором".

```
private: System::Void Form1_MouseMove(System::Object^  sender,
                                      System::Windows::Forms::MouseEventArgs^  e) {
 //цель - не координаты мыши, а точка левее и выше их на половину размеров картинки
 mousePoint = Point(e->Location.X - img->Height / 2, e->Location.Y - img->Width / 2);
}
```

Это всё, приложение можно собирать.

2. Картинка двигается по форме нажатием клавиш со стрелками

Начнём как и в прошлый раз, в пустом проекте Windows Forms к классу формы добавлены следующие свойства:

```
Image ^img; //картинка
Point imgPoint; //точка вывода картинки
Keys keyCode; //направление движения (код клавиши)
```

В конструкторе формы они проинициализированы после стандартного вызова InitializeComponent():

```
this->DoubleBuffered = true;
img = Bitmap::FromFile(Application::StartupPath + "\\butterfly.png");
imgPoint =  Point(0, 0);
```

Картинка расположена так же, как в пункте 1.

После загрузки формы точно так же создаётся таймер и программно назначается обработчик его события:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
 Timer ^ timer1 = gcnew Timer();
 timer1->Interval = 5;
 timer1->Enabled = true;
 timer1->Tick += gcnew EventHandler(this, &Form1::timer1_Tick);
}

private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs ^e)  {
 this->Invalidate();
}
```

На отрисовку теперь будем менять координаты картинки и затем уже обновлять форму:

```
private: System::Void Form1_Paint(System::Object^  sender,
                                  System::Windows::Forms::PaintEventArgs^  e) {
 switch (keyCode) {
  case Keys::Right: if (imgPoint.X+img->Width<this->ClientSize.Width) imgPoint.X ++;
  break;
  case Keys::Left: if (imgPoint.X>0) imgPoint.X --;
  break;
```

```
 case Keys::Up: if (imgPoint.Y>0) imgPoint.Y --;
 break;
 case Keys::Down: if (imgPoint.Y+img->Height<this->ClientSize.Height) imgPoint.Y ++;
 break;
 }
 e->Graphics->DrawImage(img, imgPoint);
}
```

По нажатию клавиш со стрелками будем запоминать, что нажато, в свойстве keyCode, чтобы при следующей отрисовке поменять координаты картинки на экране:

```
private: System::Void Form1_KeyDown(System::Object^  sender,
                                    System::Windows::Forms::KeyEventArgs^  e) {
 switch (e->KeyCode) {
  case Keys::Right:
  case Keys::Left:
  case Keys::Up:
  case Keys::Down:
   keyCode = e->KeyCode;
  break;
  //другие клавиши игнорируются
 }
}
```

3. Анимация на основе ImageList и таймера

Теперь это выполнится совсем просто. Опишем в классе формы данные:

```
int index, count; //номер картинки и число картинок
ImageList ^ imageList1; //компонента для списка картинок
PictureBox ^ pictureBox1; //компонента для вывода 1 картинки
```

Положим в папку Debug проекта (при выборе конфигурации решения, равной Debug) картинки с именами 1.png, ..., 6.png, обозначающие фазы движения (прикреплены внизу), и запрограммируем событие загрузки формы:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
 imageList1 = gcnew ImageList ();
 index = 0;
 count = 6;
 for (int i=1; i<=count; i++)
  imageList1->Images->Add (Image::FromFile(Application::StartupPath + "\\"+i+".png"));
 pictureBox1 = gcnew PictureBox();
 pictureBox1->Size = System::Drawing::Size
                (imageList1->Images[0]->Width,imageList1->Images[0]->Height);
```

```
pictureBox1->Image = imageList1->Images[index];
this->Location = Point (0,0);
this->Controls->Add (pictureBox1);
Timer ^ timer1 = gcnew Timer();
timer1->Interval = 100;
timer1->Enabled = true;
timer1->Tick += gcnew EventHandler(this, &Form1::timer1_Tick);
}
```

Теперь всё, что нам понадобится - выводить новую картинку по событию таймера:

```
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs ^e)  {
index++;
if (index==count) index=0;
pictureBox1->Image = imageList1->Images[index];
}
```

4. Движение картинки мышью

Так как у Image нет свойства Location, будем перемещать добавленный на форму PictureBox. Ему автоматически присвоится имя pictureBox1.

Опишем в классе формы необходимые данные:

```
Image ^ img; //картинка
int i_mouse; //флажок, показывающий, нажата ли кнопка мыши для перетаскивания
int pozx, pozy; //координаты курсора мыши на картинке
```

В конструкторе формы создадим и настроим соответствующий объект (код после вызова метода InitializeComponent()):

```
i_mouse = 0;
pictureBox1->Location = Point(0, 0);
pictureBox1->AutoSize = true;
img = Bitmap::FromFile(Application::StartupPath + "\\butterfly.png");
pictureBox1->Image = img;
pozx = pictureBox1->Width / 2;
pozy = pictureBox1->Height / 2;
this->Controls->Add(pictureBox1);
```

Для перемещения объекта мышью понадобится совместная обработка нескольких событий от мыши для компоненты pictureBox1.

По нажатию кнопки мыши будем включать режим перетаскивания картинки, а по отпусканию - выключать:

```
private: System::Void pictureBox1_MouseDown(System::Object^  sender,
                                System::Windows::Forms::MouseEventArgs^  e) {
i_mouse = 1;
}
```

```
private: System::Void pictureBox1_MouseUp(System::Object^  sender,
                         System::Windows::Forms::MouseEventArgs^  e) {
 i_mouse = 0;
}
```

Обработчик события MouseMove выполнит основную работу:

```
private: System::Void pictureBox1_MouseMove(System::Object^  sender,
                         System::Windows::Forms::MouseEventArgs^  e) {
 int mouseX = e->Location.X;
 int mouseY = e->Location.Y;
 if (i_mouse == 1 &&
    (Math::Abs(mouseX - pozx) >= 5 || Math::Abs(mouseY - pozy) >= 5)) {
  pictureBox1->Left += mouseX - pictureBox1->Width / 2;
  pictureBox1->Top += mouseY - pictureBox1->Height/ 2;
  pozx = mouseX; pozy = mouseY;
 }
}
```

"Допуск" в 5 пикселов позволяет картинке быть не слишком чувствительной к смещению мыши.

Аналогично можно реализовать перетаскивание группы объектов, например, поместив их в контейнер System::Collections::Generic::List и программно назначив всем картинкам списка одни и те же обработчики событий мыши (см. пример программного назначения события для таймера). Обработчики событий смогут различать, от какой именно картинки "пришло" событие, например, с помощью следующего кода:

```
String ^ s = ((PictureBox ^)sender)->Name->ToString();
 //вернёт имя компоненты, например, "pictureBox1"
```