TEMA 17

Cambiamos la función Draw del círculo en Elements.h

```
virtual void Draw(Graphics^ g) override
{
   g->TranslateTransform(safe_cast<float>(position.X), safe_cast<float>(position.Y));
   g->DrawEllipse(pen, 0, 0, width, height);
   g->ResetTransform();
}
```

Variamos el elemento línea

```
Line(Color color, Point start, Point end)
{
   pen = gcnew Pen(color);
   this->color = color;
   position = start;
   this->end = end - Size(start);
   boundRect =...
   ...
}
```

Y también cambiamos la función Draw de la línea

```
virtual void Draw(Graphics^ g) override
{
   g->TranslateTransform(safe_cast<float>(position.X), safe_cast<float>(position.Y));
   g->DrawLine(pen, 0, 0, end.X, end.Y);
   g->ResetTransform();
}
```

Cambiamos la función Draw del retángulo

```
virtual void Draw(Graphics^ g) override
{
   g->TranslateTransform(safe_cast<float>(position.X), safe_cast<float>(position.Y));
   g->DrawRectangle(pen, 0, 0, width, height);
   g->ResetTransform();
}
```

Y por último cambiamos la función Draw de la curva

```
virtual void Draw(Graphics^ g) override
{
   g->TranslateTransform(safe_cast<float>(position.X), safe_cast<float>(position.Y));
   Point previous(0,0);
   for each (Point p in points)
   {
      g->DrawLine(pen, previous, p);
      previous = p;
   }
   g->ResetTransform();
}
```

Se crea un nuevo archivo de encabezado, Sketch.h (Explorador de Soluciones -> botón derecho sobre nuestro proyecto -> Agregar -> Nuevo elemento -> (en el grupo Código) Archivo de encabezado (.h) -> nombre Sketch.h)

Le añadimos el siguiente código

```
// Sketch.h
// Defines a sketch

#pragma once

#include <cliext/list>
#include "Elements.h"

using namespace System;
using namespace cliext;

namespace CLRSketcher (cambiar a nuestro nombre)
{
   public ref class Sketch
   {
      private:
         list<Element^>^ elements;
      public:
      Sketch()
      {
         elements = gcnew list<Element^>();
      }
      // Add an element to the sketch
      Sketch^ operator +=(Element^ element)
      {
         elements->push_back(element);
         return this;
      }
      // Remove an element from the sketch
      Sketch^ operator -=(Element^ element)
      {
         elements->remove(element);
         return this;
      }
      void Draw(Graphics^ g)
      {
         for each (Element^ element in elements)
         element->Draw(g);
      }
   };
}
```

Añadimos una nueva variable a Form1.h

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	Sketch^	sketch	gcnew Sketch()

Y no olvidemos añadir `#include "Sketch.h"` al principio de Form1.h

Y variamos la función MouseUp de Form1.h

```
private: System::Void Form1_MouseUp(System::Object^ sender,
                                    System::Windows::Forms::MouseEventArgs^ e)
{
   if (!drawing)
      return;
   if (tempElement)
   {
      sketch += tempElement;
      tempElement = nullptr;
      Invalidate();
   }
   drawing = false;
}
```

Y también la función Paint

```
private: System::Void Form1_Paint(System::Object^ sender,
                                  System::Windows::Forms::PaintEventArgs^ e)
{
   Graphics^ g = e->Graphics;
   sketch->Draw(g);
   if...
   ...
}
```

Añadimos `public: property bool highlighted;` y `protected: Color highlightColor;` a la clase Element en Elements.h, además de incluir en la zona pública el constructor `Element() : highlightColor(Color::Magenta) { highlighted = false; }` y la propiedad `property System::Drawing::Rectangle bound {System::Drawing::Rectangle get() { return boundRect; }}`

```
public ref class Element abstract
{
   protected:
      ...
      Color highlightColor;
   public:
      ...
      property bool highlighted;
      Element() : highlightColor(Color::Magenta)
      {
         highlighted = false;
      }
      property System::Drawing::Rectangle bound
      {
         System::Drawing::Rectangle get()
         {
            return boundRect;
         }
      }
};
```

Añadimos `pen->Color = highlighted ? highlightColor : color;` a la primera línea de las funciones Draw de todos los elementos

Se añade otra función pública `bool Hit(Point p) { return boundRect.Contains(p); }` a la clase Element

Añadimos la función pública a Sketch.h

```
Element^ HitElement(Point p)
{
   list<Element^>::reverse_iterator::ReverseBidirectionalIterator riter;
   for (auto riter = elements->rbegin(); riter != elements->rend(); ++riter)
   {
      if ((*riter)->Hit(p))
         return *riter;
   }
   return nullptr;
}
```

Añadimos a Form1.h la variable

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	Element^	highlightedElement	nullptr

Y lo siguiente a la función MouseMove

```
private: System::Void Form1_MouseMove(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (drawing)
   {
      if (tempElement)
         Invalidate(tempElement->bound); // Invalidate old element area
      switch (elementType)
      {
         ...
      }
      Invalidate(tempElement->bound); // Invalidate new element area
      Update(); // Repaint
   }
   else
   {
      // Find the element under the cursor, if any
      Element^ element(sketch->HitElement(e->Location));
      if (highlightedElement == element) // If the old is same as the new
         return; // there's nothing to do
      // Reset any existing highlighted element
      if (highlightedElement)
      {
         Invalidate(highlightedElement->bound); // // Invalidate element area
         highlightedElement->highlighted = false;
         highlightedElement = nullptr;
      }
      // Find and set new highlighted element, if any
      highlightedElement = element;
      if (highlightedElement)
      {
         highlightedElement->highlighted = true;
         Invalidate(highlightedElement->bound); // Invalidate element area
      }
      Update();// Send a paint message
   }
}
```

Añadimos al final de las funciones Line, Rectangle y Circle de Elements.h

```
int penWidth(safe_cast<int>(pen->Width)); // Pen width as an integer
boundRect.Inflate(penWidth, penWidth);
```

Y un poco distinto para la función Curve

```
int width = maxX - minX;
int height = maxY - minY;
boundRect = System::Drawing::Rectangle(minX, minY, width, height);
int penWidth(safe_cast<int>(pen->Width)); // Pen width as an integer
boundRect.Inflate(penWidth, penWidth);
```

Y también la función Add de Curve

```
void Add(Point p)
{
   points->push_back(p - Size(position));
   // Modify the bounding rectangle to accommodate the new point
   int penWidth(safe_cast<int >(pen->Width)); // Pen width as an integer
   boundRect.Inflate(-penWidth, -penWidth); // Reduce by the pen width // puede dar problemas
   if...                                                              // borrar y restaurar
   {
      ...
   }
   boundRect.Inflate(penWidth, penWidth); // Inflate by the pen width
}
```

Ahora crearemos un ContextMenu: Form1.h Diseño -> Cuadro de Herramientas -> ContextMenuStrip (en las propiedades de Form1.h, indicamos que ContextMenuStrip es contextMenuStrip1 para que aparezca al hacer click en el botón derecho). Al menú contextual le añadimos las opciones `Send to Back`, `Delete` y `Move`, y cambiamos su propiedad (name) por `sendToBackContextMenuItem`, etc, respectivamente

También añadimos un separador, los cuatro elementos y los cuatro colores, y les cambiamos el nombre por `lineContextMenuItem`, etc, respectivamente, y `contextSeparator` para el separador

Hacemos doble click en los eventos Click de los botones nuevos (SendToBack, Delete y Move), para crear sus manejadores, y los demás (elementos y colores) los referimos a los que ya existen desde el MenuStrip

Para diferenciar qué menú contextual aparece recurrimos al evento Opening del propio menú contextual, según el cursor esté sobre un elemento o no, para así mostrar operaciones de modificación (SendToBack, Delete y Move) o de creación (elementos y colores) respectivamente

```
private: System::Void contextMenuStrip1_Opening(System::Object^ sender,
                                                System::ComponentModel::CancelEventArgs^ e)
{
   contextMenuStrip1->Items->Clear();// Remove existing items
   if (highlightedElement)
   {
      contextMenuStrip1->Items->Add(moveContextMenuItem);
      contextMenuStrip1->Items->Add(deleteContextMenuItem);
      contextMenuStrip1->Items->Add(sendToBackContextMenuItem);
   }
   else
   {
      contextMenuStrip1->Items->Add(lineContextMenuItem);
      contextMenuStrip1->Items->Add(rectangleContextMenuItem);
      contextMenuStrip1->Items->Add(circleContextMenuItem);
      contextMenuStrip1->Items->Add(curveContextMenuItem);
      contextMenuStrip1->Items->Add(contextSeparator);
      contextMenuStrip1->Items->Add(blackContextMenuItem);
      contextMenuStrip1->Items->Add(redContextMenuItem);
      contextMenuStrip1->Items->Add(greenContextMenuItem);
      contextMenuStrip1->Items->Add(blueContextMenuItem);
      // Set checks for the menu items
      lineContextMenuItem->Checked = elementType == ElementType::LINE;
      rectangleContextMenuItem->Checked = elementType == ElementType::RECTANGLE;
      circleContextMenuItem->Checked = elementType == ElementType::CIRCLE;
      curveContextMenuItem->Checked = elementType == ElementType::CURVE;
      blackContextMenuItem->Checked = color == Color::Black;
      redContextMenuItem->Checked = color == Color::Red;
```

```
        greenContextMenuItem->Checked = color == Color::Green;
        blueContextMenuItem->Checked = color == Color::Blue;
    }
}
```

Función Delete

```
private: System::Void deleteContextMenuItem_Click(System::Object^ sender,
                                                  System::EventArgs^ e)
{
    if (highlightedElement)
    {
        sketch -= highlightedElement; // Delete the highlighted element
        Invalidate(highlightedElement->bound);
        highlightedElement = nullptr;
        Update();
    }
}
```

Función SendToBack

```
private: System::Void sendToBackContextMenuItem_Click(System::Object^ sender,
                                                      System::EventArgs^ e)
{
    if (highlightedElement)
    {
        sketch -= highlightedElement; // Delete the highlighted element
        sketch->push_front(highlightedElement); // Then add it back to the beginning
        highlightedElement->highlighted = false;
        Invalidate(highlightedElement->bound);
        highlightedElement = nullptr;
        Update();
    }
}
```

Para lo que necesitamos la función pública `push_front` en Sketch.h

```
void push_front(Element^ element)
{
    elements->push_front(element);
}
```

Función Move, para la que tenemos que definir variables modales como `enum class Mode {Normal, Move};` al principio de Form1.h, justo después de las que definimos para ElementType

Creamos una variable para ello

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	Mode	mode	Mode::Normal

```
private: System::Void moveContextMenuItem_Click(System::Object^ sender,
                                                System::EventArgs^ e)
{
    mode = Mode::Move;
}
```

Ahora hay que modificar las funciones MouseDown, MouseMove y MouseUp, que regulan cuándo se está moviendo un elemento

```
private: System::Void Form1_MouseDown(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (e->Button == System::Windows::Forms::MouseButtons::Left)
   {
      if (mode == Mode::Normal)
      {
         drawing = true;
      }
      firstPoint = e->Location;
   }
}
```

Creamos la función pública Move en la clase Element

```
void Move(int dx, int dy)
{
   position.Offset(dx, dy);
   boundRect.X += dx;
   boundRect.Y += dy;
}
```

```
private: System::Void Form1_MouseMove(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (drawing)
   {
      if (tempElement)
         Invalidate(tempElement->Bound); // Invalidate old element area
      switch (elementType)
      {
         // Code to create a temporary element as before...
      }
      Invalidate(tempElement->bound); // The new element region
      Update();
   }
   else if (mode == Mode::Normal)
   {
      // Code to highlight the element under the cursor as before
   }
   else if (mode == Mode::Move && e->Button == System::Windows::Forms::MouseButtons::Left)
   {
      // Move the highlighted element
      if (highlightedElement)
      {
         Invalidate(highlightedElement->bound); // Region before move
         highlightedElement->Move(e->X - firstPoint.X, e->Y - firstPoint.Y);
         firstPoint = e->Location;
         Invalidate(highlightedElement->bound); // // Region after move
         Update();
      }
   }
}
```

```
private: System::Void Form1_MouseUp(System::Object^ sender,
                                    System::Windows::Forms::MouseEventArgs^ e)
{
   if (!drawing)
   {
      mode = Mode::Normal;
      return;
   }
   if (tempElement)
   {
      sketch += tempElement;
      tempElement = nullptr;
      Invalidate();
   }
   drawing = false;
}
```

Aspecto del programa al finalizar el Tema 17