USING PROBLEMS TO INTRODUCE SOME TOOLS AND TECHNOLOGIES FOR IMPLEMENTATION OF GRAPHICS AND ANIMATION IN C#

Stefka Aneva

Abstract. *This paper is devoted to the learning of event programming by using Visual C# in specialized training in Informatics in high schools. Some basic tools and technologies for the implementation of graphics and animation in C# are discussed. Two example problems are proposed.*

Keywords: event-driven programming, Graphical User Interface, element, event, event procedure, graphics, animation
2010 Mathematics Subject Classification: 97Q60

1. Introduction

Studying event programming in the specialized training in Informatics in high schools by an appropriately selected set of learning tasks with different degrees of complexity allows students to learn basic principles and possibilities of visual programming and to acquire fundamental technologies and mechanisms of implementing programs driven by events with user-friendly graphical user interface.

During the process of studying the topic "Event-driven programming in graphical user interface environment" the following five main themes are considered [1]: Basic concepts of event programming and graphical user interface (GUI); Building blocks of programming language, GUI objects; Interaction with OS; Link with databases (DB).

2. Main types of problems for the module "Event-driven programming in graphical user interface environment"

The following five types of problems must be discussed in teaching the module "Event-driven programming in graphical user interface environment":

1. The first type of problems includes a set of graphical user interface (GUI) exercises and work with the basic elements thereof.
2. The second type of problems aims at building knowledge and skills for the implementation of graphics and animation.
3. The third one is dedicated to work with arrays of GUI elements and use of special elements – GUI containers if grouping of several elements by functionality is necessary.
4. The fourth type of problems considers the implementation of user-friendly applications for databases linkage.
5. The fifth type of problems includes the creation of standard Windows applications containing menus.

3. Problems used to teach knowledge and skills for implementing graphics and animation

The resources of .NET Framework to build graphical user interfaces are defined in the namespaces System.Windows.Forms and System.Drawing (Fig. 1).

The System.Windows.Forms namespace classes and types provide tools for working with windows, dialogs, elements for text input, components selection, menus, toolbars, tables, trees, etc. The System.Windows.Forms.Design namespace contains classes which maintain the configuration of components and define the behaviour of Windows Forms controls during design. The classes and types of the System.Drawing namespace and its subspaces provide access to the Windows GDI functions: surfaces for drawing, working with graphics and image transformations, drawing geometrical shapes, working with images, texts and fonts, etc. [2]

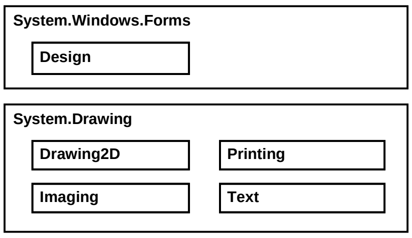


Fig. 1

The System.Drawing namespace provides a variety of basic classes, such as surfaces, pencils, brushes, classes for displaying text. The System.Drawing.Imaging namespace provides classes for working with images, classes for writing in different file formats and for image resizing. The System.Drawing.Drawing2D namespace provides classes for graphics transformations - blends, matrices and so on. The System.Drawing.Text namespace provides classes for accessing fonts of the graphic environment. The System.Drawing.Printing namespace provides classes for printer use and system dialog boxes for printing.

In the field of Information technology the term "animation" is used to describe the process in which static objects get dynamic and visually appealing appearance. To create a simple animation in a C# application, the following facilities can be used:

1. Change the location of the visible element of the GUI;
2. Show and hide elements of the GUI;
3. Create invisible elements of GUI in which in the design mode appropriate images for the Image property are chosen. During the implementation these properties are copied in the same property of another visible element.
4. Use the Windows GDI+ System.Drawing package for implementing smooth animation of a geometric object.

In each of the cases above it is important to measure the time. For this purpose, the .Net component Timer (Stopwatch) can be used. This element has no visible graphic image on the GUI of an application. It is used to raise an event at user-defined intervals, namely, it has the Interval property, which determines the time, in milliseconds, before the Tick event is generated in relation to the last occurrence of the **Tick** event (Enabled = True).

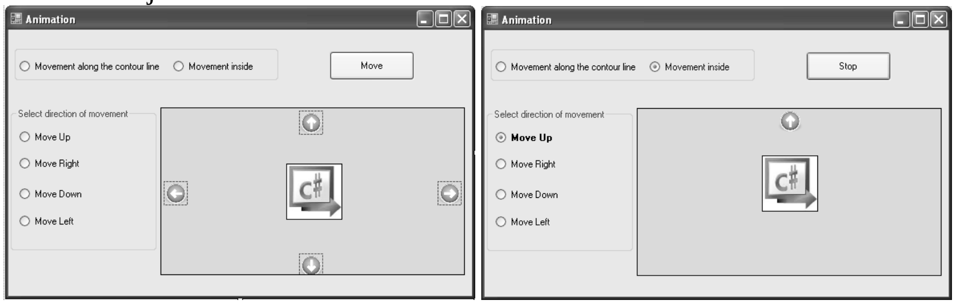
Task 1. Create a C# application which contains the following elements of the GUI (Fig. 2):

- four optional buttons implementing a selection of the desired direction of movement of an image;
- two optional buttons showing the corresponding position of a moving object (either along the contour or inside the element – container);
- one Panel element (which plays the role of container for the remaining five elements PictureBox);
- five PictureBox elements. Initially, the four images showing the arrows of the various movement directions are invisible. At a certain time only one of those images is visible depending on the chosen movement direction by an optional button.
- one button which starts and stops the movement of an image in a selected direction.
- Timer (stopwatch).

The following features should be implemented in the application:

1. In movement mode for an object, the corresponding optional button indicating the direction of movement differs from the others, as its text should be displayed in a bold style font.
2. The direction of movement can be changed as follows:
   – by pressing one of the four optional buttons setting the direction of movement;
   – by pressing the arrow keys on the keyboard.
3. The moving element can be scaled in execution mode by pressing the keys P (increase size) and M (decrease size) from the keyboard.
4. Upon reaching one of the limits of the Panel container, the image movement automatically continues clockwise along the contour as the image (arrow) indicating the direction of movement is visualized at the same time.
5. The movement of the image stops when the button which started the movement is pressed again.

Name the Project - Animation.



Design mode Fig. 2 *Execution mode*

Main objectives:

- Students get acquainted with some of the facilities in C# used to generate animations;

- To understand the purpose and get an insight of some of the properties of the .Net component – Timer discussed in the task;
- To get familiar with the purpose and the specific use of the Panel element as a container for other elements of the GUI;
- To understand the need of GUI elements – containers for grouping in the form of application which include groups of radio buttons providing different functionality;
- To reflect the need to manage the focus of a GUI element;
- To get familiar with the technology of creating and calling standard procedures in C#.
- To consolidate their knowledge of the use of multiple choice switch operator.

The following solution is proposed:

1) **<u>Stage One:</u>** <u>Creating the GUI of an application.</u>
2) **<u>Stage Two:</u>** <u>Setting some properties of GUI elements in a design mode.</u>

Table 1

GUI Elements	Name	Properties
Form	**Form1**	Text="Animation" ; KeyPreview=true
Timer	**timer1**	Interval=100
GroupBox	**groupBox1**	Text="Select direction of movement"
GroupBox	**groupBox2**	Text=""
RadioButton	**MoveUp, MoveRight MoveDown, MoveLeft**	Text="Move Up"; Text=" Move Right"; Text=" Move Down" Text=" Move Left"
RadioButton	**MoveKontur**	Text="Movement along the contour line"
RadioButton	**MoveFill**	Text="Movement inside"
Button	**buttonMove**	Text="Move"
Panel	**panel1**	
PictureBox	**PictureBox1**	SizeMode=StretchImage; Image–choice of graphic
PictureBox	**picUp, picRight, picDown, picLeft**	Image – choice of graphics

3) **<u>Stage Three:</u>** <u>Adding source code to the elements.</u>

Since selection of the desired direction of movement by using the optional buttons and by pressing the arrow buttons of the keyboard must be implemented in the application, the following requirements must be considered:

a) When setting the properties of the form in design mode, a TRUE value should be assigned to the KeyPreview property. This allows the form to handle the keyboard events before transferring them to the focused GUI element. If the value is False, each keyboard event will be handled first by the active element.
b) In execution mode, it is necessary to select the moving object pictureBox1 as an active (focused) element whenever you click on an element from the interface of the application, for which an Event procedure is generated. Otherwise, by using the arrow buttons of the keyboard it will not be possible to directly change the direction of movement but navigation among the different elements of the GUI on the form will be performed instead.

✓ Implementing the action of the timer.

```
int posoka;
private void timer2_Tick(object sender, EventArgs e)
  {  switch (posoka)
      { case 1:
            if (buttonMove.Text == "Stop") MoveUp.Font = new Font(MoveUp.Font, FontStyle.Bold);
            if (pictureBox1.Top >= 0) pictureBox1.Location = new Point(pictureBox1.Left, pictureBox1.Top - 1);
            else  MoveRight.Checked = true; break;
        case 2:  if (buttonMove.Text == "Stop") MoveRight.Font = new Font(MoveRight.Font, FontStyle.Bold);
            if (pictureBox1.Left + pictureBox1.Width <= panel1.Width-2)
                pictureBox1.Location = new Point(pictureBox1.Left + 1, pictureBox1.Top);
            else MoveDown.Checked = true; break;
        case 3:
            if (buttonMove.Text == "Stop") MoveDown.Font = new Font(MoveDown.Font, FontStyle.Bold);
            if (pictureBox1.Top + pictureBox1.Height < panel1.Height )
                pictureBox1.Location = new Point(pictureBox1.Left , pictureBox1.Top+1);
            else MoveLeft.Checked = true; break;
        case 4:
            if (buttonMove.Text == "Stop") MoveLeft.Font = new Font(MoveLeft.Font, FontStyle.Bold);
            if (pictureBox1.Left >=0)  pictureBox1.Location = new Point(pictureBox1.Left - 1, pictureBox1.Top);
            else MoveUp.Checked = true; break; }
        if ((pictureBox1.Left == 0)||(pictureBox1.Top == 0)
            || (pictureBox1.Left + pictureBox1.Width >= panel1.Width-1)
            || (pictureBox1.Top + pictureBox1.Height >= panel1.Height))
         { MoveKontur.ForeColor = Color.Blue; MoveFill.ForeColor = Color.Black; MoveKontur.Checked = true; }
         else
            if ((pictureBox1.Left > 0) && (pictureBox1.Top > 0)
            && (pictureBox1.Left + pictureBox1.Width < panel1.Width)
            && (pictureBox1.Top + pictureBox1.Height < panel1.Height))
            { MoveKontur.ForeColor = Color.Black; MoveFill.ForeColor = Color.Blue; MoveFill.Checked = true; } }
```

✓ Implementing the event procedure **Form1_Load** for initial initialization.

```
private void Form1_Load(object sender, EventArgs e)
      {   pictureBox1.Left = panel1.Width / 2 - pictureBox1.Width / 2;
          pictureBox1.Top = panel1.Height / 2 - pictureBox1.Height / 2;
          picUp.Left = panel1.Width / 2 - picUp.Width / 2; picUp.Top = 0;
          picRight.Left = panel1.Width - picUp.Width; picRight.Top = panel1.Height / 2 - picUp.Height/2;
          picDown.Left = panel1.Width / 2 - picDown.Width / 2; picDown.Top = panel1.Height - picDown.Height;
          picLeft.Left = 0; picLeft.Top = panel1.Height / 2 - picLeft.Height / 2;
          MoveUp.Checked = true; MoveFill.Checked = true; groupBox1.Focus();}
```

✓ Implementing the standard procedures **button_clearbold** and **hide_picarrow**.

```
void button_clearbold()
   { MoveUp.Font = new Font(MoveUp.Font, FontStyle.Regular);
     MoveRight.Font = new Font(MoveRight.Font, FontStyle.Regular);
     MoveDown.Font = new Font(MoveDown.Font, FontStyle.Regular);
     MoveLeft.Font = new Font(MoveLeft.Font, FontStyle.Regular);}
void hide_picarrow() {picUp.Visible = false;picRight.Visible = false;picDown.Visible = false;picLeft.Visible = false;}
```

✓ Implementing the action of the optional buttons setting the direction of movement.

```
private void Up_CheckedChanged(object sender, EventArgs e)
  { posoka = 1; pictureBox1.Focus(); hide_picarrow(); picUp.Visible = true;
    if (timer2.Enabled == true)  { button_clearbold(); MoveUp.Font = new Font(MoveUp.Font, FontStyle.Bold); }}
private void Right_CheckedChanged(object sender, EventArgs e)
 {posoka = 2; pictureBox1.Focus(); hide_picarrow();picRight.Visible = true;
  if (timer2.Enabled == true) { button_clearbold(); MoveRight.Font = new Font(MoveRight.Font, FontStyle.Bold);}}
private void Down_CheckedChanged(object sender, EventArgs e)
 {posoka = 3; pictureBox1.Focus(); hide_picarrow(); picDown.Visible = true;
  if (timer2.Enabled == true) {button_clearbold(); MoveDown.Font = new Font(MoveDown.Font, FontStyle.Bold);}}
private void Left_CheckedChanged(object sender, EventArgs e)
 { posoka = 4; pictureBox1.Focus(); hide_picarrow(); picLeft.Visible = true;
   if (timer2.Enabled == true) { button_clearbold(); MoveLeft.Font = new Font(MoveLeft.Font, FontStyle.Bold);}}
```

✓ Implementing the action of the button **buttonMove**.

```
private void buttonMove_Click(object sender, EventArgs e)
  { pictureBox1.Focus(); if (buttonMove.Text == "Move") { timer2.Enabled = true; buttonMove.Text = "Stop"; }
    else if (buttonMove.Text == "Stop") { timer2.Enabled = false; buttonMove.Text = "Move"; button_clearbold();} }
```

✓ Implementing the action of the event procedure **Form1_KeyDown**.

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
  { if (e.KeyCode == Keys.Up) { MoveUp.Checked = true; pictureBox1.Focus(); }
    if (e.KeyCode == Keys.Right) { MoveRight.Checked = true; pictureBox1.Focus(); }
    if (e.KeyCode == Keys.Down) { MoveDown.Checked = true; pictureBox1.Focus(); }
    if (e.KeyCode == Keys.Left) { MoveLeft.Checked = true; pictureBox1.Focus(); }
    if (e.KeyCode == Keys.P)
      { pictureBox1.Size = new Size(pictureBox1.Width + 10, pictureBox1.Height + 10); pictureBox1.Focus(); }
    if (e.KeyCode == Keys.M)
      { pictureBox1.Size = new Size(pictureBox1.Width - 10, pictureBox1.Height - 10); pictureBox1.Focus(); } }
```

✓ Implementing the action of the radio buttons showing the position of a moving element in the container.

```
private void MoveKontur_CheckedChanged(object sender, EventArgs e)  {  pictureBox1.Focus(); }
private void MoveFill_CheckedChanged(object sender, EventArgs e)      {  pictureBox1.Focus(); }
```

Task 2. Create a C# application which contains the following GUI elements (Fig.3):

- five optional buttons implementing a selection of the desired geometric shape or image for drawing;
- a checkbox, which sets whether the geometric shape will be filled in or not;
- one PictureBox element and three command buttons.
- Timer and three colorDialog components for colour selection.

The following features should be implemented in the application:

1. Upon reaching one of the boundaries of the PictureBox container, the moving image's motion must automatically continue in the opposite direction.
2. The change of the fill colour and the pen colour of the geometric shape which is selected for drawing, as well as the text colour, should be performed through the three command buttons.

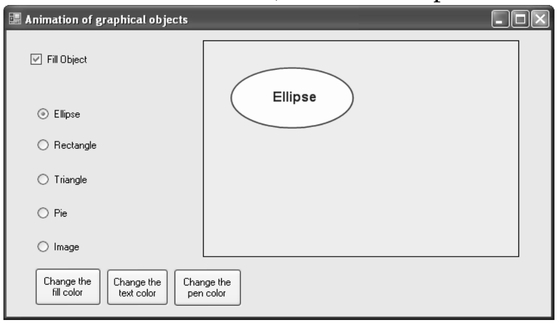


Fig. 3

Name the project - Graphic.

<u>Main objectives:</u>

- The students become familiar with some mechanisms for graphics realization in C# and they acquire the following main skills: working with some basic methods for drawing graphic objects; brushes setup, pencils for drawing, work with texts and fonts;
- To know the purpose and specific use of the element PictureBox as a container;
- To deepen their knowledge and skills for the implementing animation and the use of the Timer element;
- To understand the purpose and to get an insight of some of the properties of the discussed in the task colorDialog component.

The following solution is proposed:

1) **Stage One:** Creating the GUI of an application.

In this problem we shall use the **PictureBox** element as a container, since it does not draw anything in its Paint method, which is called before each redraw. If Panel or any other element is used, a flicker will appear.

2) **Stage Two:** Setting some properties of GUI elements in a design mode.

Table 2

GUI Elements	Name	Properties
Form	**Form1**	Text="Animation of graphical objects"
Timer	**timer1**	
RadioButton	**radioButton1, radioButton2 radioButton3, radioButton4 radioButton5**	Text="Ellipse"; Text="Rectangle" Text="Triangle"; Text="Pie" Text="Image"
PictureBox	**picDraw**	
CheckBox	**checkBox_Fill**	Text="Fill Object"
Button	**button_changeFillColor**	Text="Change the fill color"
Button	**button_changeTextColor**	Text="Change the text color"
Button	**button_changePenColor**	Text="Change the pen color"
colorDialog	**colorDialog1, colorDialog2 colorDialog3**	

3) **Stage Three:** Adding source code to the elements.

```
int t_X = 0; int t_Y = 0; int s = 1; int w_X = 140; int h_Y = 70; int p =1; Boolean p_Fill;
Brush G_brush = new SolidBrush(Color.LightYellow); Pen G_pen = new Pen(Color.Red,3);
Brush G_text = new SolidBrush(Color.Blue); Font font = new Font("Arial", 12, FontStyle.Bold);
Image Image = Image.FromFile("d:/pic.jpg");
```

✓ Implementing the action of the timer.

```
private void timer1_Tick(object sender, EventArgs e)
  { t_X = t_X + s; t_Y = t_Y+s;
   if ((t_X <= 0) ||(t_Y <= 0)||(t_X + w_X >= picDraw.Width -5)||(t_Y + h_Y >= picDraw.Height -5))  { s = -s;}
   picDraw.Refresh(); }
```

✓ Implementing the action of the event procedure **picDraw_Paint**.

```
private void picDraw_Paint(object sender, PaintEventArgs e)
 {Graphics g = e.Graphics; g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
  switch (p)
   {case 1:
     g.DrawEllipse(G_pen, t_X, t_Y, w_X, h_Y); if (p_Fill == true) g.FillEllipse(G_brush, t_X, t_Y, w_X, h_Y);
     g.DrawString("Ellipse", font, G_text, t_X + 45, t_Y + 25); break;
    case 2:
     g.DrawRectangle(G_pen, t_X, t_Y, w_X, h_Y);
     if (p_Fill == true) g.FillRectangle(G_brush, t_X, t_Y, w_X, h_Y);
     g.DrawString("Rectangle", font, G_text, t_X + 28, t_Y + 25); break;
    case 3:
     Point point1 = new Point(t_X + w_X / 2, t_Y); Point point2 = new Point(t_X, t_Y + h_Y);
     Point point3 = new Point(t_X + w_X, t_Y + h_Y); Point[] curvePoints = { point1, point2, point3 };
     g.DrawPolygon(G_pen, curvePoints); if (p_Fill == true) g.FillPolygon(G_brush, curvePoints);
     g.DrawString("Triangle", font, G_text, t_X + 36, t_Y + 50); break;
    case 4:
     Rectangle rect = new Rectangle(t_X, t_Y, w_X / 2, w_X / 2);
     float startAngle =  0.0F; float sweepAngle = 360.0F; g.DrawPie(G_pen, rect, startAngle, sweepAngle);
     if (p_Fill == true) g.FillPie(G_brush, rect, startAngle, sweepAngle);
     g.DrawString("Pie", font, G_text, t_X + 22, t_Y + 25); break;
    case 5:  g.DrawImage(Image, t_X, t_Y); g.DrawString("Image", font, G_text, t_X+30, t_Y + h_Y/2); break;}}
```

✓ Implementing the action of the radio buttons for selection of the drawing object.

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)    { p = 1; w_X = 140; h_Y = 70; }
private void radioButton2_CheckedChanged(object sender, EventArgs e)    { p = 2; w_X = 140; h_Y = 70; }
private void radioButton3_CheckedChanged(object sender, EventArgs e)   { p = 3; w_X = 140; h_Y = 70; }
private void radioButton4_CheckedChanged(object sender, EventArgs e)   { p = 4; w_X = 140; h_Y = 70; }
private void radioButton5_CheckedChanged(object sender, EventArgs e)
  { p = 5; w_X = Image.Width; h_Y = Image.Height;}
```

✓ Implementing the action of the buttons for selection of the fill colour, text colour and pen colour.

```
private void button_changeFillColor_Click(object sender, EventArgs e)
  { if (colorDialog1.ShowDialog() == DialogResult.OK)
     { G_brush.Dispose(); G_brush = new SolidBrush(colorDialog1.Color);}}
private void button_changeTextColor_Click(object sender, EventArgs e)
  { if (colorDialog2.ShowDialog() == DialogResult.OK)
     { G_text.Dispose(); G_text = new SolidBrush(colorDialog2.Color); }}
private void button_changePenColor_Click(object sender, EventArgs e)
  { if (colorDialog3.ShowDialog() == DialogResult.OK)
     { G_pen.Dispose(); G_pen = new Pen(colorDialog3.Color,3); }}
```

✓ Implementing the action of the checkbox, which sets whether the geometric object will be filled in or not.

```
private void checkBox_Fill_CheckedChanged(object sender, EventArgs e)
  { if (checkBox_Fill.CheckState == CheckState.Checked) p_Fill = true; else p_Fill = false;}
private void Form1_Load(object sender, EventArgs e)  { checkBox_Fill.Checked = true;}
```

4. Conclusion

The aims of this stage are as follows:

– the skills for proper selection of suitable elements of the GUI in accordance with the required functionality [3,4] of the GUI application should be further developed;
– the main mechanisms and technologies for the implementation of animation should be refined;
– the purpose of the Timer component should be clear and its use when implementing animation of a GUI element or a graphic object in the form of application must be well understood;
– the basic functions of GDI+ on Windows, accessible via the System.Drawing namespace should be acquired, as well as the basic methods for drawing graphic objects, namely line, ellipse, rectangle, polygon, curve, etc.; work with images, text and fonts;
– the need for GUI container elements should be well understood;
– abilities to create and call standard procedures in C# applications should exist.

References

[1] MOMN, Directorate "Politics in general education". Syllabus part III for basic and specialized training – IX, X, XI and XII class. Sofia, 2003.
[2] S. Nakov, et al. Programming for .Net Framework. Volume 2, Bars, 2007.
[3] N. Valchanov, T. Terzieva, V. Shkurtov, A. Iliev. Architecture of extensible computations driven systems. Proceedings of the Thirty-ninth Spring Conference of the Union of Bulgarian Mathematicians, Albena, April 6-10, 2010, 207-211.
[4] N. Valchanov, T. Terzieva, V. Shkurtov, A. Iliev. Approaches in Building and Supporting Business Information Systems. Proceedings of International Conference on Information Technology in Business Management, Varna, Oct. 16-17, 2009, 100-105.

Stefka Yordanova Aneva
Faculty of Mathematics and Informatics, University of Plovdiv “Paisii Hilendarski”
236 Bulgaria Blvd., 4003 Plovdiv, Bulgaria, e-mail: stfaneva@uni-plovdiv.bg