APARAT DE MASURA

Descrierea programului

Acest program reprezinta un aparat de masura universal. Acest apparat poate fi modificat in functie de necesitatile utilizatorului. Modificarile pe care aparatul le suporta sunt: dimensiunea, zonele de avertizare (verde, galben, rosu), numarul de marcaje si dimensiunea marcajelor.

Apasand pe butonul “Generate”, se va genera un apparat cu setari prestabilite, care pot fi modificate folosind elementele din partea dreapta a aplicatiei.

Codul sursa

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Tools
{
   public partial class Form1 : Form
   {
      Graphics draw;
      Gauge gauge;


      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         draw = CreateGraphics();
      }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            draw.Clear(this.BackColor);
            gauge = new Gauge(new Point(50, 50), new Size(200, 200));
            gauge.DrawGauge(ref draw);
        }

        // green zone procent
        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown1.Value + gauge.yellowZoneProcent + gauge.redZoneProcent >
100)
            {
                numericUpDown1.Value = 100 - gauge.yellowZoneProcent -
gauge.redZoneProcent;
            }
            gauge.greenZoneProcent = (int)numericUpDown1.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // yellow zone procent
        private void numericUpDown2_ValueChanged(object sender, EventArgs e)
        {
            if (gauge.greenZoneProcent + numericUpDown2.Value + gauge.redZoneProcent >
100)
            {
                numericUpDown2.Value = 100 - gauge.greenZoneProcent - gauge.redZoneProcent;
            }
            gauge.yellowZoneProcent = (int)numericUpDown2.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // red zone procent
        private void numericUpDown3_ValueChanged(object sender, EventArgs e)
        {
            if (gauge.greenZoneProcent + gauge.yellowZoneProcent + numericUpDown3.Value
> 100)
            {
                numericUpDown3.Value = 100 - gauge.greenZoneProcent -
gauge.yellowZoneProcent;
            }
            gauge.redZoneProcent = (int)numericUpDown3.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
```

```
        }

        // long mark length procent
        private void numericUpDown4_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown4.Value + gauge.DistanceToMarksProcent > 95)
            {
                numericUpDown4.Value = 95 - gauge.DistanceToMarksProcent;
            }
            gauge.LongMarkLengthProcent = (int)numericUpDown4.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // short mark length procent
        private void numericUpDown5_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown5.Value + gauge.DistanceToMarksProcent > 95)
            {
                numericUpDown5.Value = 95 - gauge.DistanceToMarksProcent;
            }
            gauge.ShortMarkLengthProcent = (int)numericUpDown5.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // distance to marks procent
        private void numericUpDown6_ValueChanged(object sender, EventArgs e)
        {
            if (numericUpDown6.Value + gauge.LongMarkLengthProcent > 95)
            {
                numericUpDown6.Value = 95 - gauge.LongMarkLengthProcent;
            }
            gauge.DistanceToMarksProcent = (int)numericUpDown6.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            gauge.needleRad = gauge.beginMarkInRad + Math.PI / 2 -
ExtraMath.Procent(trackBar1.Value / 2, gauge.markGridRads);
            label10.Text = "Value: " + ExtraMath.Procent(trackBar1.Value / 2, gauge.marks *
gauge.shortMarks);
            if (gauge.needleRad < gauge.endMarkInRad + Math.PI / 2)
                gauge.needleRad = gauge.endMarkInRad + Math.PI / 2;
            gauge.SetProperties();
```

```
            gauge.DrawNeedle(ref draw);
        }

        // marks
        private void numericUpDown7_ValueChanged(object sender, EventArgs e)
        {
            gauge.marks = (int)numericUpDown7.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // short marks
        private void numericUpDown8_ValueChanged(object sender, EventArgs e)
        {
            gauge.shortMarks = (int)numericUpDown8.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // grid angle
        private void numericUpDown9_ValueChanged(object sender, EventArgs e)
        {
            gauge.markGridAngle = (float)numericUpDown9.Value;

            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }

        // gauge size
        private void numericUpDown12_ValueChanged(object sender, EventArgs e)
        {
            gauge.gaugeSize = new Size((int)numericUpDown12.Value,
(int)numericUpDown12.Value);
            draw.Clear(this.BackColor);
            gauge.SetProperties();
            gauge.DrawGauge(ref draw);
        }
    }

    public class Coords
    {
        public int x;
        public int y;

        public static Coords operator +(Coords a, Coords b)
        {
            return new Coords(a.x + b.x, a.y + b.y);
        }
```

```
    public static explicit operator Coords(Point p)
    {
        Coords c = new Coords(p.X, p.Y);
        return c;
    }
    public static Coords Left = new Coords(0, -1);
    public static Coords Right = new Coords(0, 1);
    public static Coords Down = new Coords(1, 0);
    public static Coords Up = new Coords(-1, 0);

    public Coords(int _x, int _y)
    {
        this.x = _x;
        this.y = _y;
    }
    public Coords(double _x, double _y)
    {
        this.x = (int)_x;
        this.y = (int)_y;
    }

    public static int DistanceBetween(Coords a, Coords b)
    {
        return (int)Math.Sqrt(Math.Pow(a.x - b.x, 2) + Math.Pow(a.y - b.y, 2));
    }
    public Point ToPoint()
    {
        return new Point(this.x, this.y);
    }
}

public class Object
{

}

public class Gauge
{
    public Size gaugeSize;
    public Point gaugePos;
    Rectangle gaugeRect;
    Point gaugeCenter;

    #region dial design
    public int rimWidth = 5;
    public Color rimColor = Color.Black;
    public Color backColor = Color.Gray;

    public int greenZoneProcent = 60;
    public Color greenZoneColor = Color.Green;
```

```
        double greenZoneRad;
        double startGreenZone;

        public int yellowZoneProcent = 20;
        public Color yellowZoneColor = Color.Yellow;
        double yellowZoneRad;
        double startYellowZone;

        public int redZoneProcent = 20;
        public Color redZoneColor = Color.Red;
        double redZoneRad;
        double startRedZone;

        public int LongMarkLengthProcent = 20;
        public int ShortMarkLengthProcent = 10;
        public int DistanceToMarksProcent = 70;



        public Color dialColor;
        public int dialSize;
        public Color dial2Color;
        public int dial2Size;
        #endregion

        public double needleLength;
        public int needleWidth = 3;
        public double needleRad = 2.3;
        Point needleTip;
        public Color needleColor = Color.Black;

        public Color markColor = Color.Black;

        #region gauge properties
        public int maxVal; // maximum value displayed in gauge
        public int marks = 20; // the number of marks displayed on the mark grid
        public int shortMarks = 5;
        public float markGridAngle = 180;
        public double markGridRads; // the range of the mark grid
        public double beginMarkInRad; // the position in radians of the first mark in grid
        public double endMarkInRad; // the position in radians of the last mark in grid
        public int markLength; // the length of a mark in grid
        public int markDistFromCenter; // the distance from the center of the gauge to the mark
in the grid
        public int value; // the value displayed in the gauge
        public string title; // the title of the gauge (what measures the gauge)
        #endregion
        #region gauge methods

        public void ChangeValue (int _value)
```

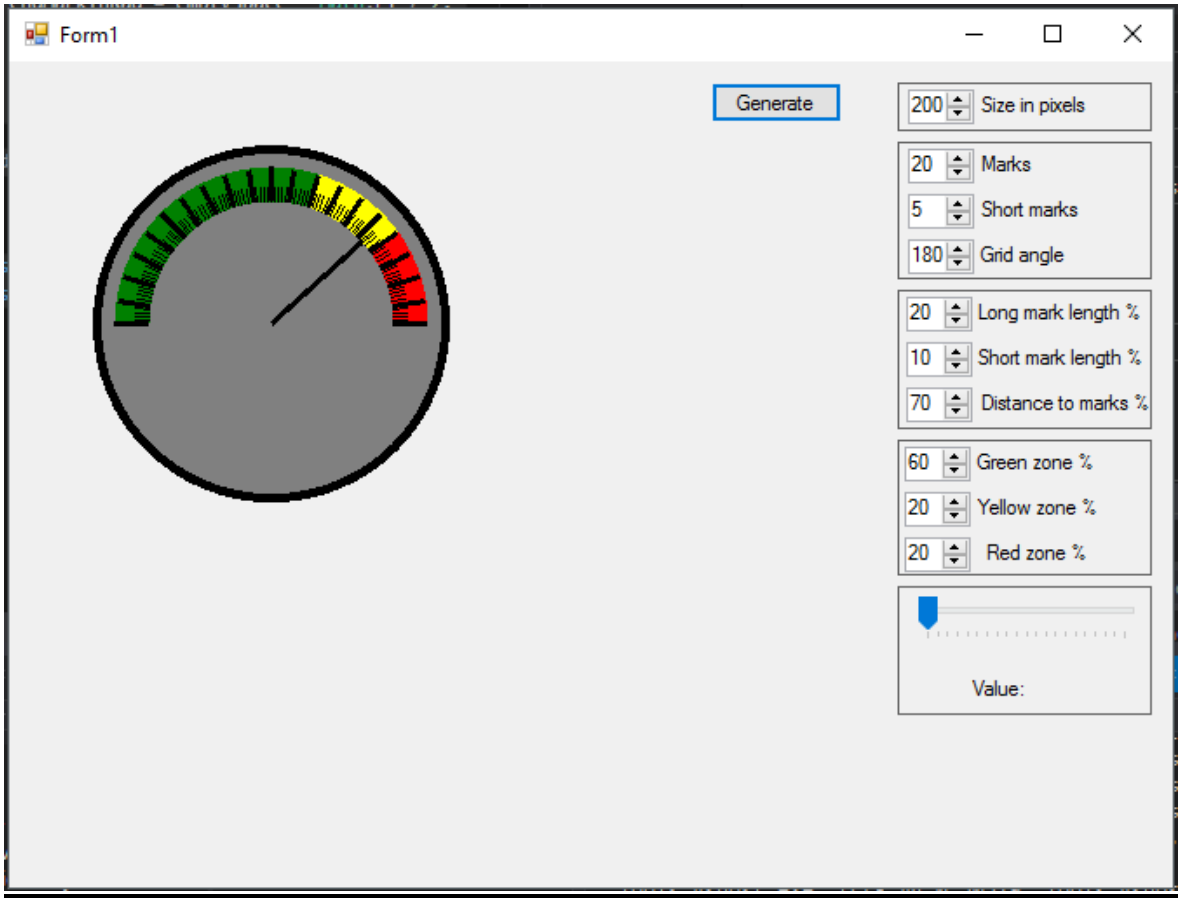
```
        {
            this.value = _value;
        }

        public void DrawGauge(ref Graphics _draw)
        {
            // background
            _draw.FillEllipse(new SolidBrush(this.backColor), new Rectangle(this.gaugePos,
this.gaugeSize));

            // warning zones
            _draw.FillPie(new SolidBrush(greenZoneColor),
ExtraMath.Procent(DistanceToMarksProcent + LongMarkLengthProcent,gaugeRect),
ExtraMath.Rad2Deg(startGreenZone), ExtraMath.Rad2Deg(greenZoneRad));
            _draw.FillPie(new SolidBrush(yellowZoneColor),
ExtraMath.Procent(DistanceToMarksProcent + LongMarkLengthProcent, gaugeRect),
ExtraMath.Rad2Deg(startYellowZone), ExtraMath.Rad2Deg(yellowZoneRad));
            _draw.FillPie(new SolidBrush(redZoneColor),
ExtraMath.Procent(DistanceToMarksProcent + LongMarkLengthProcent, gaugeRect),
ExtraMath.Rad2Deg(startRedZone), ExtraMath.Rad2Deg(redZoneRad));
            _draw.FillEllipse(new SolidBrush(this.backColor),
ExtraMath.Procent(DistanceToMarksProcent, gaugeRect));

            // gauge rim
            _draw.DrawEllipse(new Pen(this.rimColor, (float)this.rimWidth), gaugeRect);

            // draw needle
            this.needleTip = new Point((int)(Math.Sin(needleRad) * needleLength +
this.gaugeCenter.X), (int)(Math.Cos(needleRad) * needleLength + this.gaugeCenter.Y));
            _draw.DrawLine(new Pen(this.needleColor, this.needleWidth), this.gaugeCenter,
this.needleTip);

            DrawMarkGrid(ref _draw);
            DrawNeedle(ref _draw);
        }

        public void DrawNeedle(ref Graphics _draw)
        {
            _draw.FillEllipse(new SolidBrush(this.backColor),
ExtraMath.Procent(DistanceToMarksProcent, gaugeRect));
            this.needleTip = new Point((int)(Math.Sin(needleRad) * needleLength +
this.gaugeCenter.X), (int)(Math.Cos(needleRad) * needleLength + this.gaugeCenter.Y));
            _draw.DrawLine(new Pen(this.needleColor, this.needleWidth), this.gaugeCenter,
this.needleTip);
        }

        #region draw mark grid

        void DrawMarkGrid(ref Graphics _draw)
        {
```

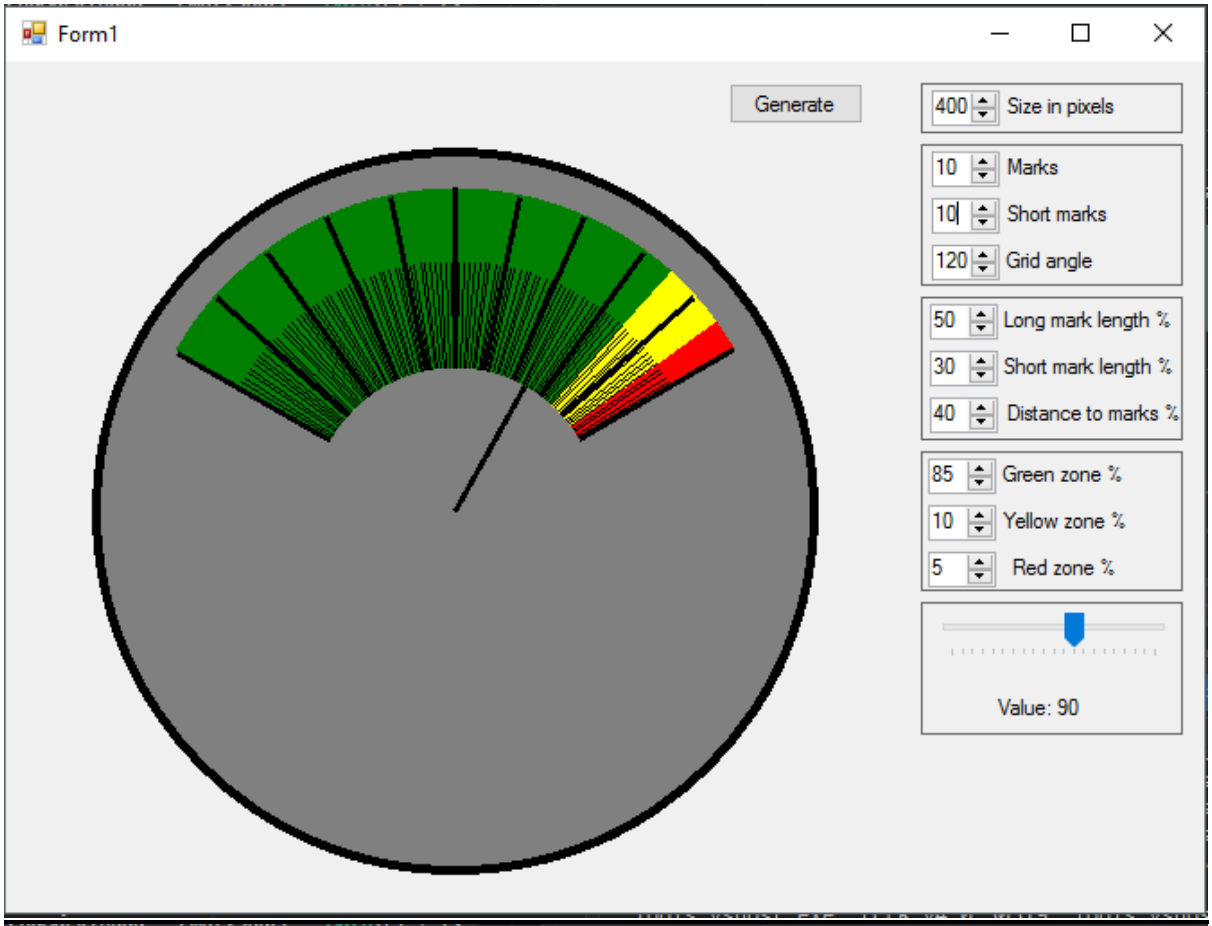
```
            double markStep = markGridRads / (double)marks;
            double pos = beginMarkInRad;
            for (int i = 0; i <= marks; i ++)
            {
                if (pos != beginMarkInRad)
                {
                    for (double shortPos = pos; shortPos <= pos + markStep; shortPos += markStep /
(double)shortMarks)
                    {
                        Point c = new Point();
                        Point d = new Point();
                        SetLineForMark(ref c, ref d, shortPos, DistanceToMarksProcent,
ShortMarkLengthProcent);
                        _draw.DrawLine(new Pen(Color.Black, 1), c, d);
                    }
                }
                Point a = new Point();
                Point b = new Point();
                SetLineForMark(ref a, ref b, pos, DistanceToMarksProcent,
LongMarkLengthProcent);
                _draw.DrawLine(new Pen(Color.Black, 3), a, b);
                pos -= markStep;
            }
        }

        void SetLineForMark(ref Point begin, ref Point end, double posInRad, int
procentFromCenter, int procentLength)
        {
            begin = new Point(gaugeCenter.X + (int)(Math.Cos(posInRad) *
ExtraMath.Procent(procentFromCenter, gaugeSize.Width / 2)), gaugeCenter.Y +
(int)(Math.Sin(posInRad + Math.PI) * ExtraMath.Procent(procentFromCenter,
gaugeSize.Width / 2)));
            end = new Point(begin.X + (int)(Math.Cos(posInRad) *
ExtraMath.Procent(procentLength, gaugeSize.Width / 2)), begin.Y +
(int)(Math.Sin(posInRad + Math.PI) * ExtraMath.Procent(procentLength, gaugeSize.Width /
2)));
        }

        #endregion

        #endregion

        #region gauge constructors

        public Gauge(Point _pos, Size _size)
        {
            this.gaugePos = _pos;
            this.gaugeSize = _size;
            SetProperties();
        }
```

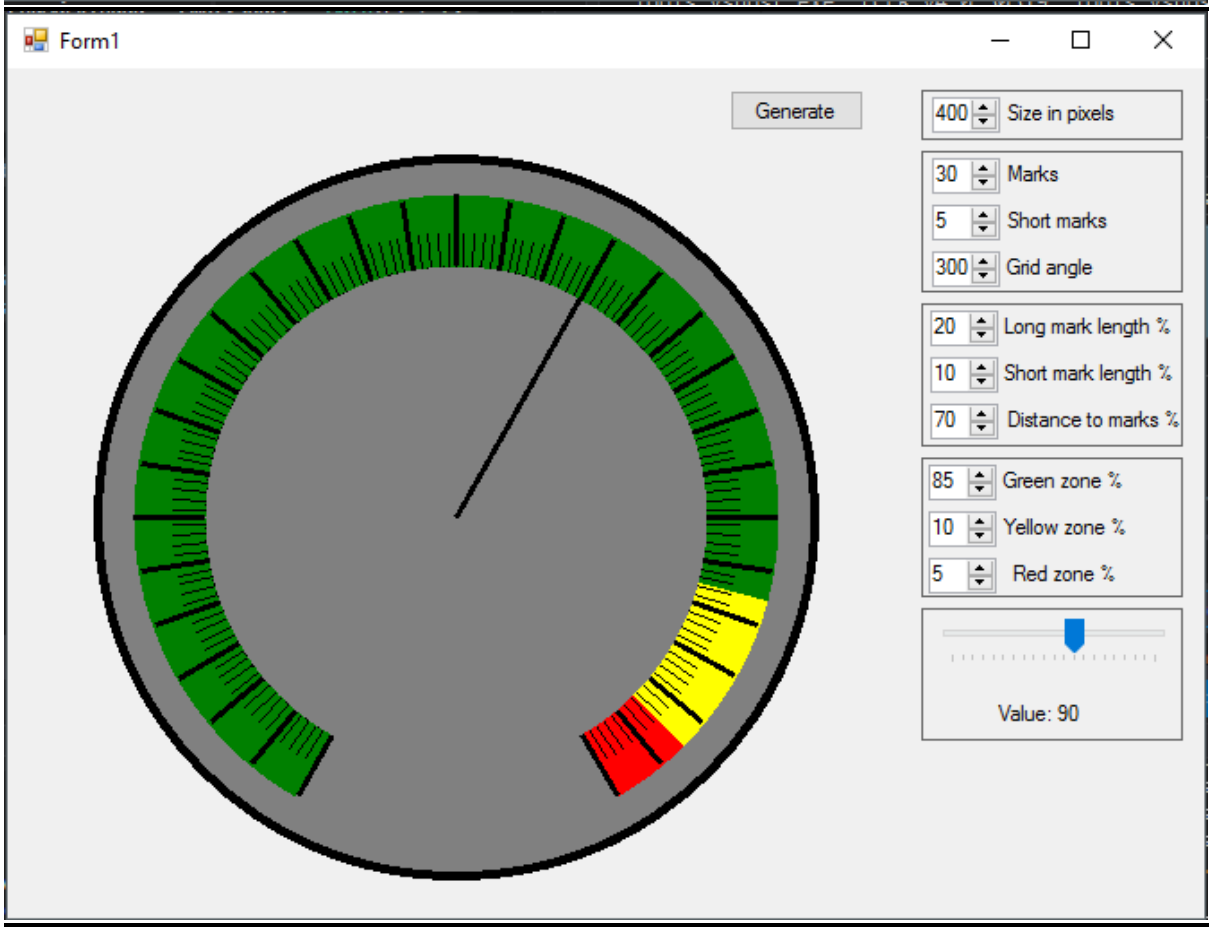
```
        public Gauge()
        {

        }

        public void SetProperties()
        {
            this.markGridRads = ExtraMath.Deg2Rad(markGridAngle);
            this.gaugeCenter = new Point(this.gaugePos.X + this.gaugeSize.Width / 2,
this.gaugePos.Y + this.gaugeSize.Height / 2);
            this.gaugeRect = new Rectangle(this.gaugePos, this.gaugeSize);
            this.greenZoneRad = ExtraMath.Procent(greenZoneProcent, markGridRads);
            this.yellowZoneRad = ExtraMath.Procent(yellowZoneProcent, markGridRads);
            this.redZoneRad = ExtraMath.Procent(redZoneProcent, markGridRads);
            double emptySpace = (2 * Math.PI - markGridRads) / 2;
            beginMarkInRad = markGridRads + emptySpace - Math.PI / 2;
            endMarkInRad = emptySpace - Math.PI / 2;
            this.startGreenZone = beginMarkInRad + Math.PI - markGridRads;
            this.startYellowZone = startGreenZone + greenZoneRad;
            this.startRedZone = startYellowZone + yellowZoneRad;
            this.needleLength = ExtraMath.Procent(DistanceToMarksProcent, gaugeSize.Width /
2);
        }
        #endregion

    }

    public static class Debug
    {
        public static void Log(string output)
        {
            Console.WriteLine(output);
        }
    }
    public static class ExtraMath
    {
        public static float Rad2Deg(double radians)
        {
            return (float)(radians * 180 / Math.PI);
        }
        public static double Deg2Rad(float angle)
        {
            return (double)(angle * Math.PI / 180);
        }

        public static int Procent(int procent, int value)
        {
            return (int)(value * procent / 100f);
        }
```

```
        public static float Procent(float procent, float value)
        {
            return (float)(value * procent / 100f);
        }
        public static double Procent(double procent, double value)
        {
            return (double)(value * procent / 100);
        }
        public static Rectangle Procent(int procent, Rectangle rect)
        {
            Point center = new Point(rect.Location.X + rect.Size.Width / 2, rect.Location.Y +
rect.Size.Height / 2);
            return MakeRectAroundCenter(center, new Size((int)(rect.Width * procent / 100f),
(int)(rect.Height * procent / 100f)));
        }

        public static Rectangle MakeRectAroundCenter(Point center, Size size)
        {
            return new Rectangle(new Point(center.X - size.Width / 2, center.Y - size.Height / 2),
size);
        }
    }
}
```

Rularea aplicatiei




Form1
Generate
400 Size in pixels
10 Marks
10 Short marks
120 Grid angle
50 Long mark length %
30 Short mark length %
40 Distance to marks %
85 Green zone %
10 Yellow zone %
5 Red zone %
Value: 90


Form1
Generate
400 Size in pixels
30 Marks
5 Short marks
300 Grid angle
20 Long mark length %
10 Short mark length %
70 Distance to marks %
85 Green zone %
10 Yellow zone %
5 Red zone %
Value: 90