**Вариант** **11** ((459175 % 15) + 1)

11. Прямоугольник вращается вокруг одной из своих вершин. Сделав полный оборот, прямоугольник перемещается с приращением по координате Х. Внутри прямоугольника движется точка.

**Программная реализация**

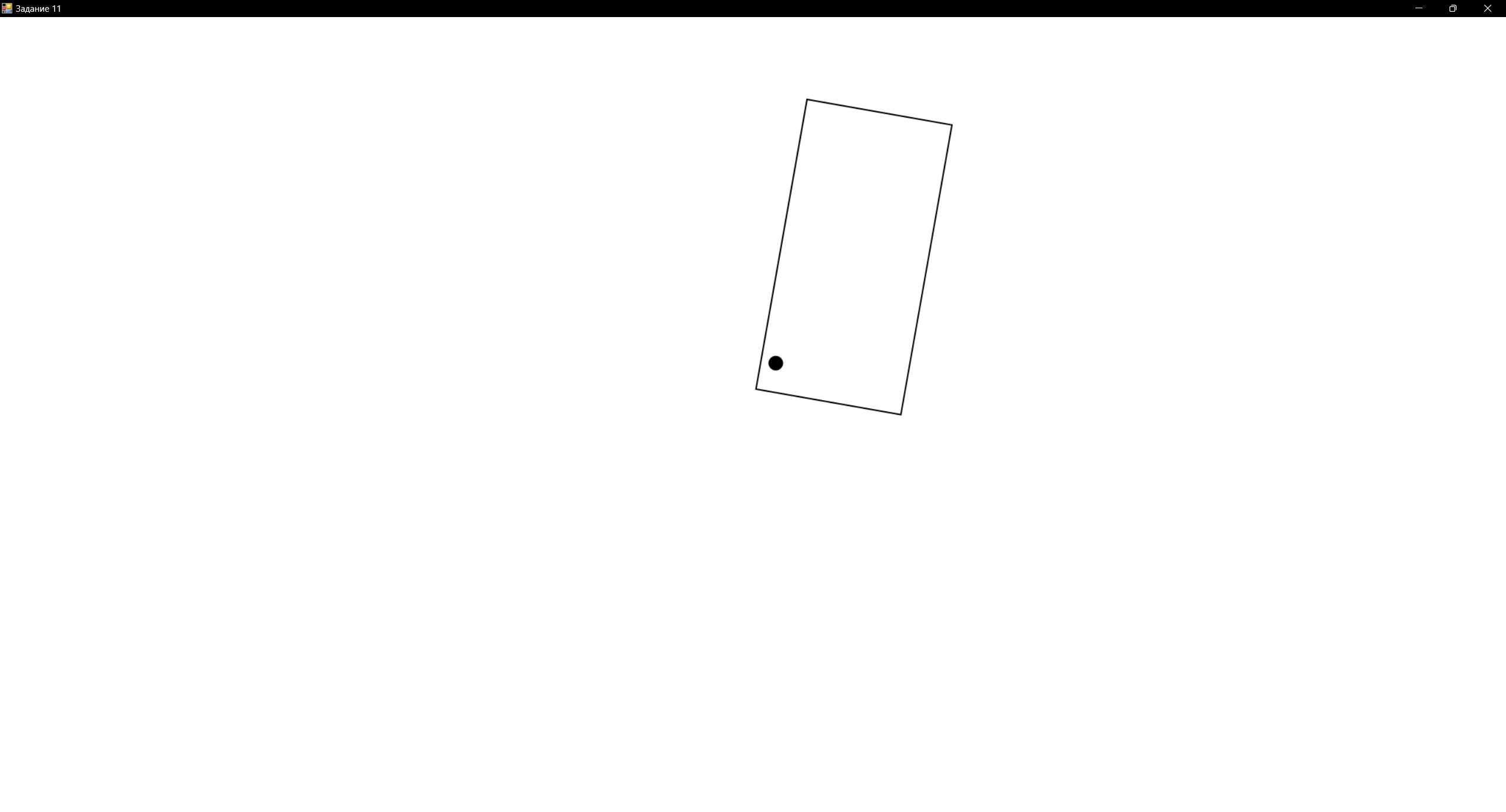
using System;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Windows.Forms;  
  
namespace task11  
{  
 public partial class MainForm : Form  
 {  
 private readonly Graphics Graph;  
 private readonly Pen MyPen;  
 private readonly SolidBrush MyBrush;  
 private Timer timerRectangle, timerPoint;  
   
 private GraphicsPath rectangle;  
 private PointF[] points;  
 private bool freeX = true, freeY = true;  
 private int xPoint, yPoint;  
  
 private int centerY;  
 private int countRotates;  
 private bool toLeft = true;  
  
 private const float **RotateAngle** = 20;  
 private const int **PointSpeed** = 10;  
 private const int **RectangleLength** = 400;  
 private const int **RectangleHeight** = 200;  
  
 public MainForm()  
 {  
 InitializeComponent();  
 Graph = CreateGraphics();  
 Graph.SmoothingMode = SmoothingMode.**HighQuality**;  
 MyPen = new Pen(Color.Black, 2);  
 MyBrush = new SolidBrush(Color.Black);  
  
 InitTimerPoint(50);  
 InitTimerRectangle(300);  
 InitRectangle();  
 InitPoint();  
 }  
  
 private void InitTimerPoint(int interval)  
 {  
 timerPoint = new Timer();  
 timerPoint.Interval = interval;  
 timerPoint.Enabled = true;  
 timerPoint.Tick += timerPoint\_tick;  
 }  
   
 private void InitTimerRectangle(int interval)  
 {  
 timerRectangle = new Timer();  
 timerRectangle.Interval = interval;  
 timerRectangle.Enabled = true;  
 timerRectangle.Tick += timerRectangle\_tick;  
 }  
  
 private void InitRectangle()  
 {  
 countRotates = 0;  
 centerY = ClientSize.Height / 2;  
 points = new PointF[4];  
 points[0] = new PointF(ClientSize.Width / 2 - **RectangleLength** / 2, ClientSize.Height / 2);  
 points[1] = new PointF(ClientSize.Width / 2 + **RectangleLength** / 2, ClientSize.Height / 2);  
 points[2] = new PointF(ClientSize.Width / 2 + **RectangleLength** / 2, ClientSize.Height / 2 + **RectangleHeight**);  
 points[3] = new PointF(ClientSize.Width / 2 - **RectangleLength** / 2, ClientSize.Height / 2 + **RectangleHeight**);  
 }  
  
 private void InitPoint()  
 {  
 xPoint = ClientSize.Width / 2;  
 yPoint = ClientSize.Height / 2 + **RectangleHeight** / 2;  
 }  
  
 private void timerRectangle\_tick(object sender, EventArgs e)  
 {  
 if (countRotates == (int)(360 / **RotateAngle**))  
 {  
 OffsetX();  
 countRotates = 0;  
 }  
  
 UpdateScreen();  
 Rotate();  
 countRotates++;  
 }  
  
 private void OffsetX()  
 {  
 if (toLeft && points[3].X - 50 < 0)  
 {  
 toLeft = false;  
 }  
  
 if (!toLeft && points[0].X + 50 > ClientSize.Width)  
 {  
 toLeft = true;  
 }  
   
 for (var i = 0; i < points.Length; i++)  
 {  
   
 var curX = toLeft ? points[i].X - 50 : points[i].X + 50;  
 var curY = points[i].Y;  
 points[i] = new PointF(curX, curY);  
 }  
  
 rectangle = new GraphicsPath();  
 rectangle.AddPolygon(points);  
 }  
  
 private void Rotate()  
 {  
 const double **angleRadian** = **RotateAngle** \* Math.**PI** / 180;  
 var pointRotate = points[1];  
 for (var i = 0; i < points.Length; i++)  
 {  
 var curX = (float)((points[i].X - pointRotate.X) \* Math.Cos(**angleRadian**) - (points[i].Y - pointRotate.Y) \* Math.Sin(**angleRadian**) + pointRotate.X);  
 var curY = (float)((points[i].X - pointRotate.X) \* Math.Sin(**angleRadian**) + (points[i].Y - pointRotate.Y) \* Math.Cos(**angleRadian**) + pointRotate.Y);  
 points[i] = new PointF(curX, curY);  
 }  
  
 rectangle = new GraphicsPath();  
 rectangle.AddPolygon(points);  
 }  
  
 private void UpdateScreen()  
 {  
 Graph.Clear(Color.White);  
 PaintRectangle();  
 PaintPoint();  
 }  
   
 private void timerPoint\_tick(object sender, EventArgs e)  
 {  
 if (rectangle == null) return;  
  
 if (freeX) xPoint += **PointSpeed**;  
 else xPoint -= **PointSpeed**;  
  
 if (freeY) yPoint += **PointSpeed**;  
 else yPoint -= **PointSpeed**;  
   
 UpdateScreen();  
  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint + **PointSpeed** + centerY / 10 + 10))) freeY = false;  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint - **PointSpeed** - 10))) freeY = true;  
 if (!rectangle.IsVisible(new Point(xPoint - **PointSpeed** - 10, yPoint))) freeX = true;  
 if (!rectangle.IsVisible(new Point(xPoint + **PointSpeed** + centerY / 10 + 10, yPoint))) freeX = false;  
 }  
   
 private void PaintRectangle()  
 {  
 Graph.DrawPolygon(MyPen, points);  
 }  
   
 private void PaintPoint()  
 {  
 Graph.FillEllipse(MyBrush, xPoint, yPoint, 20, 20);  
 }  
  
 private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  
 {  
 Graph.Dispose();  
 MyPen.Dispose();  
 timerRectangle.Dispose();  
 timerPoint.Dispose();  
 }  
 }  
}

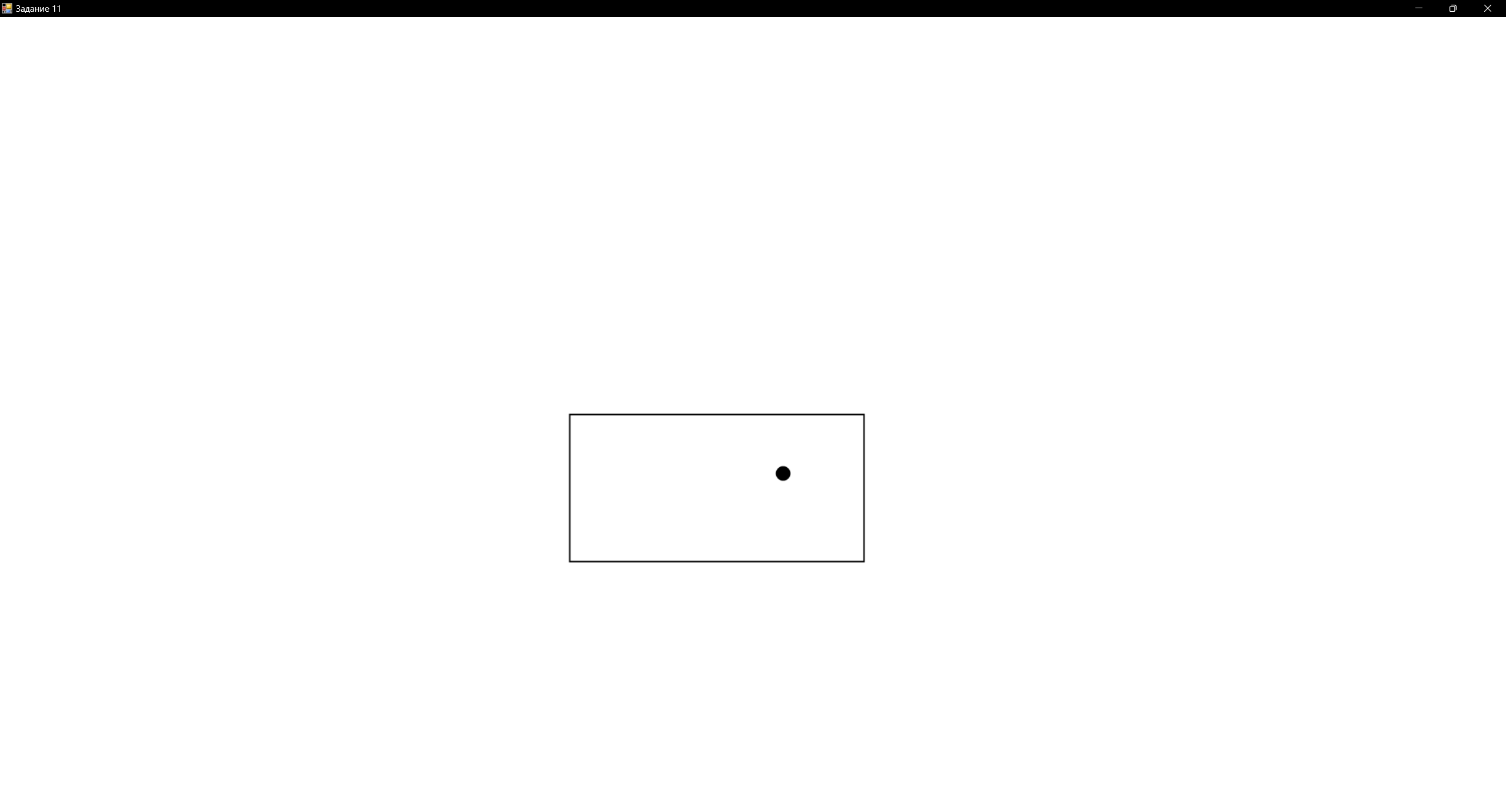
**Описание**

Прямоугольник хранится в виде массива точек. Описана функция поворота фигуры относительно своей вершины. Для определения полного оборота объявлен счетчик поворотов, который инкрементируется каждый поворот и обнуляется, когда фигура сделает полный оборот. Следуя из этого, когда счетчик обнуляется, вместо поворота фигуры осуществляется её перемещение по оси X.

Точка движется в одном направлении, пока её следующие координаты не выйдут за пределы прямоугольника, иначе она меняет направление.

**Внешний вид**





**Трассировка**

Изначально прямоугольник отрисовывается по центру формы вместе с точкой.

private void UpdateScreen()  
{  
 Graph.Clear(Color.White);  
 PaintRectangle();  
 PaintPoint();  
}

Далее считаются новые координаты фигуры с учетом поворота относительно правой верхней вершины.

private void Rotate()  
{  
 const double **angleRadian** = **RotateAngle** \* Math.**PI** / 180;  
 var pointRotate = points[1];  
 for (var i = 0; i < points.Length; i++)  
 {  
 var curX = (float)((points[i].X - pointRotate.X) \* Math.Cos(**angleRadian**) - (points[i].Y - pointRotate.Y) \* Math.Sin(**angleRadian**) + pointRotate.X);  
 var curY = (float)((points[i].X - pointRotate.X) \* Math.Sin(**angleRadian**) + (points[i].Y - pointRotate.Y) \* Math.Cos(**angleRadian**) + pointRotate.Y);  
 points[i] = new PointF(curX, curY);  
 }  
  
 rectangle = new GraphicsPath();  
 rectangle.AddPolygon(points);  
}

Форма очищается, и прямоугольник отрисовывается заново, а также увеличивается счетчик поворотов.

private void timerRectangle\_tick(object sender, EventArgs e)  
{  
 if (countRotates == (int)(360 / **RotateAngle**))  
 {  
 OffsetX();  
 countRotates = 0;  
 }  
  
 UpdateScreen();  
 Rotate();  
 countRotates++;  
}

Если количество поворотов равно количеству углов в круге, то фигура сдвигается влево. Если в следующий шаг левая точка прямоугольника выйдет за пределы экрана, то прямоугольник меняет направление движения направо.

private void OffsetX()  
{  
 if (toLeft && points[3].X - 50 < 0)  
 {  
 toLeft = false;  
 }  
  
 if (!toLeft && points[0].X + 50 > ClientSize.Width)  
 {  
 toLeft = true;  
 }  
   
 for (var i = 0; i < points.Length; i++)  
 {  
   
 var curX = toLeft ? points[i].X - 50 : points[i].X + 50;  
 var curY = points[i].Y;  
 points[i] = new PointF(curX, curY);  
 }  
  
 rectangle = new GraphicsPath();  
 rectangle.AddPolygon(points);  
}

Параллельно происходит движение точки, которое происходит в доступном направлении. Если следующий шаг точки выходит за границы прямоугольника, то точка меняет направление.

private void timerPoint\_tick(object sender, EventArgs e)  
{  
 if (rectangle == null) return;  
  
 if (freeX) xPoint += **PointSpeed**;  
 else xPoint -= **PointSpeed**;  
  
 if (freeY) yPoint += **PointSpeed**;  
 else yPoint -= **PointSpeed**;  
   
 UpdateScreen();  
  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint + **PointSpeed** + centerY / 10 + 10))) freeY = false;  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint - **PointSpeed** - 10))) freeY = true;  
 if (!rectangle.IsVisible(new Point(xPoint - **PointSpeed** - 10, yPoint))) freeX = true;  
 if (!rectangle.IsVisible(new Point(xPoint + **PointSpeed** + centerY / 10 + 10, yPoint))) freeX = false;  
}