**Змейка**

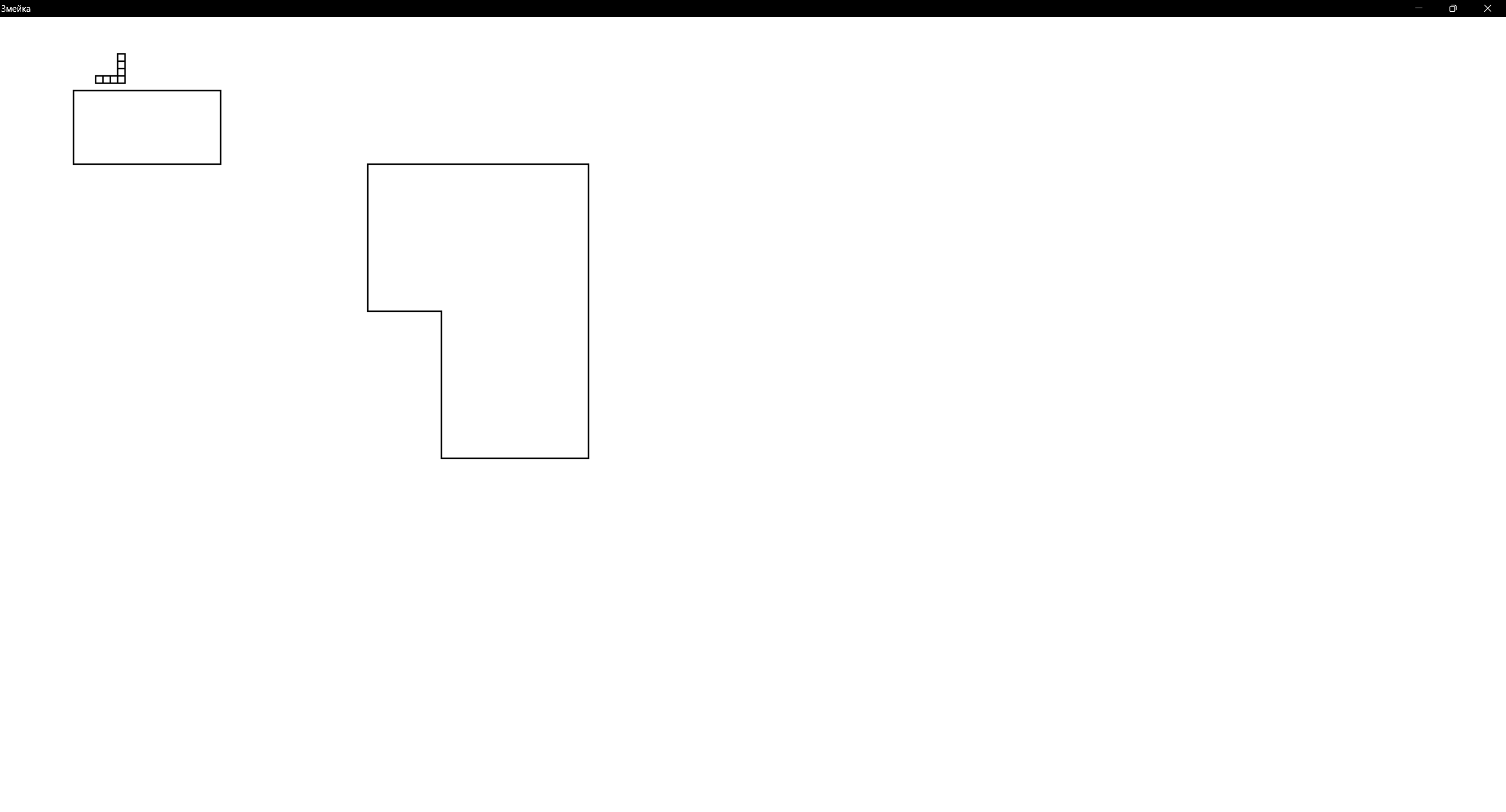
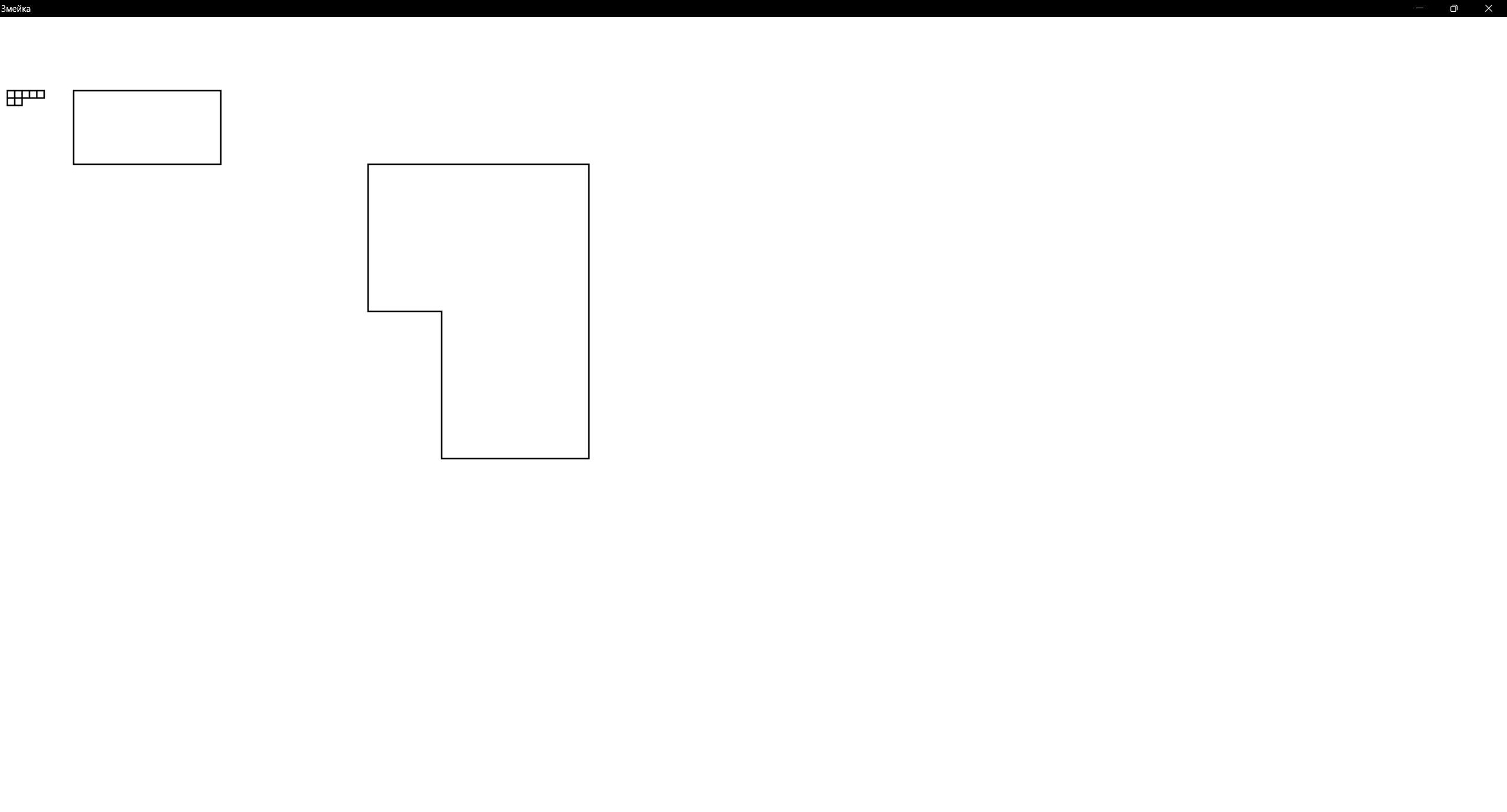
**Программная реализация**

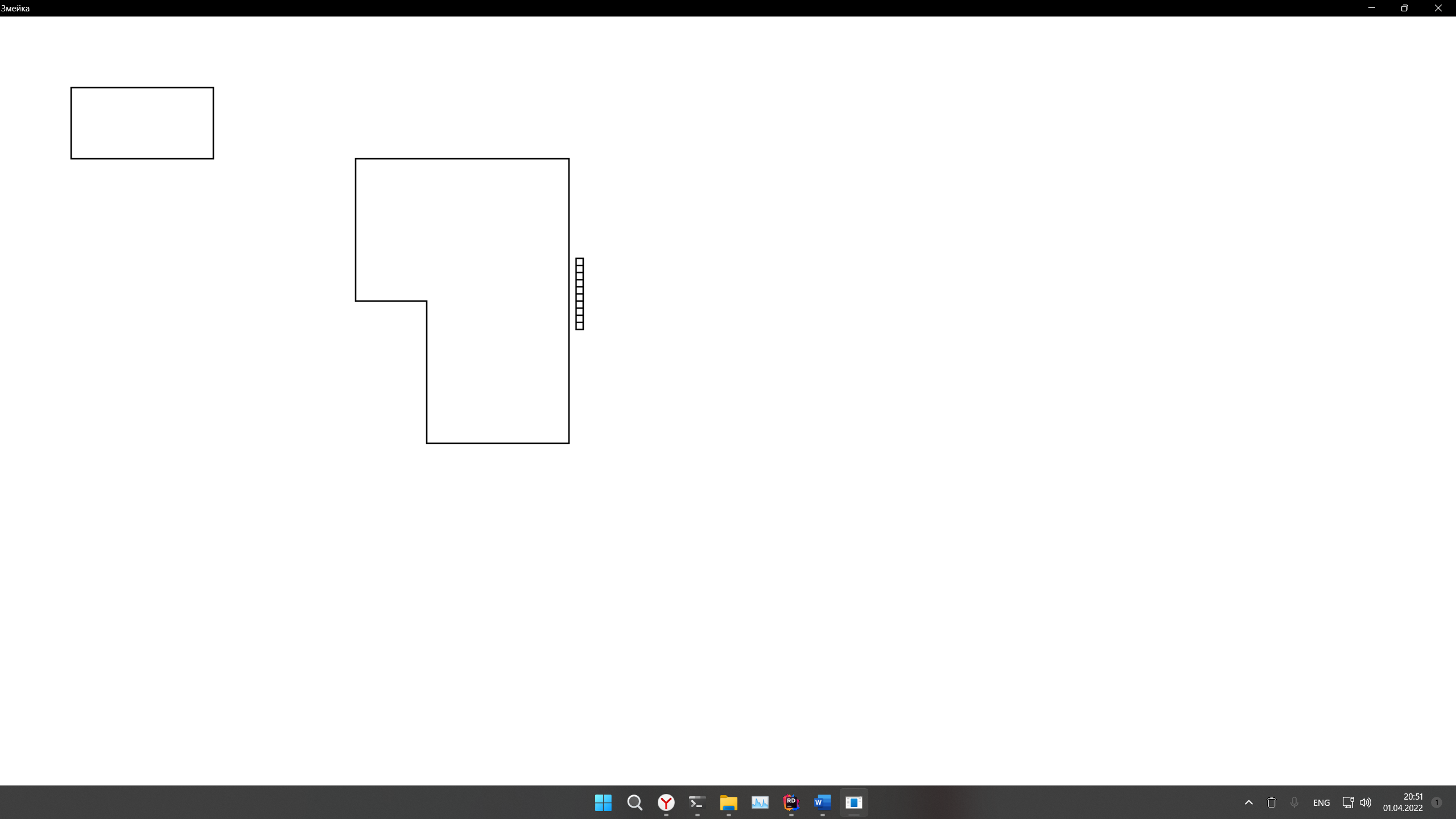
using System;  
using System.Collections.Generic;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Linq;  
using System.Windows.Forms;  
  
namespace snake  
{  
 public partial class MainForm : Form  
 {  
 private readonly Graphics Graph;  
 private readonly Pen MyPen;  
 private Timer timer;  
  
  
 private bool isConnected; *// Флаг соединения с препятствием* private int countSteps; *// Количество шагов* private PointF headCoordinates; *// Координаты головы* private PointF curConnectPoint; *// Точка текущего соединения* private enum Orientation *// Перечисление направлений змейки* {  
 **UP**, **RIGHT**, **DOWN**, **LEFT** }  
  
 private Orientation curOrientation = Orientation.**DOWN**; *// Текущее направление* private Orientation curBarrier; *// Текущая фигура* private readonly Queue<GraphicsPath> snake; *// Змейка* private readonly Dictionary<GraphicsPath, bool> barriers; *// Препятствия* private const float **SizeSegment** = 10; *// Размер сегмента змейки* private const int **Fps** = 30; *// Количество кадров в секунду* public MainForm()  
 {  
 InitializeComponent();  
 Graph = CreateGraphics();  
 Graph.SmoothingMode = SmoothingMode.**HighQuality**;  
 MyPen = new Pen(Color.Black, 2);  
  
 snake = new Queue<GraphicsPath>();  
 barriers = new Dictionary<GraphicsPath, bool>();  
  
 CreateBarriers();  
 CreateSnake(150, 0, 10);  
  
 InitTimer(1000 / **Fps**);  
 }  
   
 private void InitTimer(int interval)  
 {  
 timer = new Timer();  
 timer.Interval = interval;  
 timer.Tick += timer\_tick;  
 timer.Enabled = true;  
 }  
   
 */// <summary>  
 /// Каждый тик перерисовывает окно  
 /// </summary>  
 /// <param name="sender"></param>  
 /// <param name="e"></param>* private void timer\_tick(object sender, EventArgs e)  
 {  
 Invalidate();  
 }  
   
 private void MainForm\_Paint(object sender, PaintEventArgs e)  
 {  
 PaintBarriers(e.Graphics);  
 MoveSnake(e.Graphics);  
 PaintSnake(e.Graphics);  
 }  
  
 */// <summary>  
 /// Создание препятствий  
 /// </summary>* private void CreateBarriers()  
 {  
 var barrier = new GraphicsPath();  
   
 var points = new PointF[4];  
 points[0] = new PointF(100, 100);  
 points[1] = new PointF(300, 100);  
 points[2] = new PointF(300, 200);  
 points[3] = new PointF(100, 200);  
 barrier.AddPolygon(points);  
 barriers.Add(barrier, false);  
   
 points = new PointF[6];  
 points[0] = new PointF(500, 200);  
 points[1] = new PointF(800, 200);  
 points[2] = new PointF(800, 600);  
 points[3] = new PointF(600, 600);  
 points[4] = new PointF(600, 400);  
 points[5] = new PointF(500, 400);  
 barrier = new GraphicsPath();  
 barrier.AddPolygon(points);  
 barriers.Add(barrier, false);  
 }  
   
 */// <summary>  
 /// Создание змейки  
 /// </summary>  
 /// <param name="x">Начальная координата</param>  
 /// <param name="y">Начальная координата</param>  
 /// <param name="length">Длина змейки в сегментах</param>* private void CreateSnake(float x, float y, int length)  
 {  
 for (var i = 0; i < length; i++)  
 {  
 var sectionCoordinate = new PointF(x + **SizeSegment**, y);  
 var size = new SizeF(**SizeSegment**, **SizeSegment**);  
 var section = new RectangleF(sectionCoordinate, size);  
 var path = new GraphicsPath();  
   
 path.AddRectangle(section);  
 snake.Enqueue(path);  
 Graph.DrawPath(MyPen, path);  
   
 headCoordinates = sectionCoordinate;  
 }  
 }  
  
 */// <summary>  
 /// Отрисовка препятствий  
 /// </summary>  
 /// <param name="graph"></param>* private void PaintBarriers(Graphics graph)  
 {  
 foreach (var barrier in barriers)  
 {  
 graph.DrawPath(MyPen, barrier.Key);  
 }  
 }  
  
 */// <summary>  
 /// Считает новые координаты змейки  
 /// </summary>  
 /// <param name="graph"></param>* private void MoveSnake(Graphics graph)  
 {  
 *// Определяем следующее направление змейки* CalculateNextOrientation(graph);  
 *// Делаем шаг в этом направлении* StepSnake();  
 }  
   
 */// <summary>  
 /// Определяет следующее направление змейки  
 /// </summary>  
 /// <param name="graph"></param>* private void CalculateNextOrientation(Graphics graph)  
 {  
 var endBarrier = true;  
  
 *// Если змейка соединена с препятствием* if (isConnected)  
 {  
 var nextPoint = GetNextPoint();  
 *// Если сделан полный обход препятствия* if (countSteps > 2 && Math.Abs(nextPoint.X - curConnectPoint.X) <= **SizeSegment** && Math.Abs(nextPoint.Y - curConnectPoint.Y) <= **SizeSegment**)  
 {  
 *// Отсоединяемся* isConnected = false;  
 return;  
 }  
   
 countSteps++;  
   
 var connectPoint = GetConnectPoint();  
 foreach (var barrier in barriers.Keys)  
 {  
 *// Если впереди фигура* if (IsEntersBorder(barrier, nextPoint))  
 {  
 *// Поворачиваем* curOrientation = (Orientation)((int)(curBarrier + 2) % 4);  
 curBarrier = (Orientation)((int)(curBarrier + 1) % 4);  
 return;  
 }  
   
 *// Если сторона препятствия не закончилась* if (IsEntersBorder(barrier, connectPoint))  
 {  
 endBarrier = false;  
 }  
 }  
   
 *// Не поворачиваем* if (!endBarrier) return;  
   
 *// Иначе поворачиваем* curOrientation = curBarrier;  
 curBarrier = (Orientation)((int)(curBarrier + 3) % 4);  
 }   
 *// Если не соединены с препятствием* else  
 {  
 *// Проверяем, находится ли впереди ещё не проходимое препятствие* CheckConnect(graph);  
 }  
 }  
   
 */// <summary>  
 /// Делает шаг  
 /// </summary>  
 /// <exception cref="ArgumentOutOfRangeException"></exception>* private void StepSnake()  
 {  
 switch (curOrientation)  
 {  
 case Orientation.**UP**:  
 StepUp();  
 break;  
 case Orientation.**RIGHT**:  
 StepRight();  
 break;  
 case Orientation.**DOWN**:  
 StepDown();  
 break;  
 case Orientation.**LEFT**:  
 StepLeft();  
 break;  
 default:  
 throw new ArgumentOutOfRangeException();  
 }  
 }  
  
 */// <summary>  
 /// Получение следующей точки по направлению движения  
 /// </summary>  
 /// <returns></returns>  
 /// <exception cref="ArgumentOutOfRangeException"></exception>* private PointF GetNextPoint()  
 {  
 PointF nextPoint;  
 switch (curOrientation)  
 {  
 case Orientation.**UP**:  
 nextPoint = new PointF(headCoordinates.X, headCoordinates.Y - **SizeSegment**);  
 break;  
 case Orientation.**RIGHT**:  
 nextPoint = new PointF(headCoordinates.X + **SizeSegment**, headCoordinates.Y);  
 break;  
 case Orientation.**LEFT**:  
 nextPoint = new PointF(headCoordinates.X - **SizeSegment**, headCoordinates.Y);  
 break;  
 case Orientation.**DOWN**:  
 nextPoint = new PointF(headCoordinates.X, headCoordinates.Y + **SizeSegment**);  
 break;  
 default:  
 throw new ArgumentOutOfRangeException();  
 }  
 return nextPoint;  
 }  
   
 */// <summary>  
 /// Получение точки соединения с препятствием  
 /// </summary>  
 /// <returns></returns>  
 /// <exception cref="ArgumentOutOfRangeException"></exception>* private PointF GetConnectPoint()  
 {  
 PointF connectPoint;  
 switch (curBarrier)  
 {  
 case Orientation.**UP**:  
 connectPoint = new PointF(headCoordinates.X, headCoordinates.Y - **SizeSegment**);  
 break;  
 case Orientation.**RIGHT**:  
 connectPoint = new PointF(headCoordinates.X + **SizeSegment**, headCoordinates.Y);  
 break;  
 case Orientation.**LEFT**:  
 connectPoint = new PointF(headCoordinates.X - **SizeSegment**, headCoordinates.Y);  
 break;  
 case Orientation.**DOWN**:  
 connectPoint = new PointF(headCoordinates.X, headCoordinates.Y + **SizeSegment**);  
 break;  
 default:  
 throw new ArgumentOutOfRangeException();  
 }  
 return connectPoint;  
 }  
   
 */// <summary>  
 /// Проверяет, пересекла ли голова змеи препятствие  
 /// </summary>  
 /// <param name="path"></param>  
 /// <param name="point"></param>  
 /// <returns></returns>* private bool IsEntersBorder(GraphicsPath path, PointF point)  
 {  
 var p1 = new PointF(point.X, point.Y);  
 var p2 = new PointF(point.X, point.Y + **SizeSegment**);  
 var p3 = new PointF(point.X + **SizeSegment**, point.Y);  
 var p4 = new PointF(point.X + **SizeSegment**, point.Y + **SizeSegment**);  
  
 if (path.IsVisible(p1) || path.IsVisible(p2) || path.IsVisible(p3) || path.IsVisible(p4))  
 {  
 return true;  
 }  
   
 return path.IsOutlineVisible(p1, MyPen) || path.IsOutlineVisible(p2, MyPen) ||  
 path.IsOutlineVisible(p3, MyPen) || path.IsOutlineVisible(p4, MyPen);  
 }  
   
 */// <summary>  
 /// Проверяет, находится ли впереди ещё не проходимое препятствие  
 /// </summary>  
 /// <param name="graph"></param>* private void CheckConnect(Graphics graph)  
 {  
 var nextPoint = GetNextPoint();  
 if (nextPoint.X >= ClientSize.Width )  
 {  
 StepDown();  
 curOrientation = Orientation.**LEFT**;  
 }  
 else if (nextPoint.X <= 0)   
 {  
 StepDown();  
 curOrientation = Orientation.**RIGHT**;  
 }  
 else if(nextPoint.Y <= 0)  
 {  
 StepRight();  
 curOrientation = Orientation.**DOWN**;  
 }  
 else if (nextPoint.Y >= ClientSize.Height)  
 {  
 StepRight();  
 curOrientation = Orientation.**UP**;  
 }  
   
 PaintSnake(graph);  
  
 for (var i = 0; i < barriers.Count; i++)  
 {  
 if (!IsEntersBorder(barriers.ElementAt(i).Key, nextPoint)) continue;  
   
 if (barriers.ElementAt(i).Value)  
 {  
 StepBound(graph);  
 return;  
 }  
   
 isConnected = true;  
 countSteps = 0;  
 curConnectPoint = headCoordinates;  
  
 *// Помечаем препятствие как пройденное* var path = barriers.ElementAt(i).Key;  
 barriers.Remove(barriers.ElementAt(i).Key);  
 barriers.Add(path, true);  
  
 curBarrier = curOrientation;  
 curOrientation = (Orientation)((int)(curOrientation + 1) % 4);  
 CalculateNextOrientation(graph);  
 return;  
 }  
 }  
   
 */// <summary>  
 /// Меняет направление при сталкивании с препятствием или границей окна  
 /// </summary>  
 /// <param name="graph"></param>  
 /// <exception cref="ArgumentOutOfRangeException"></exception>* private void StepBound(Graphics graph)  
 {  
 switch (curOrientation)  
 {  
 case Orientation.**RIGHT**:  
 StepDown();  
 curOrientation = Orientation.**LEFT**;  
 break;  
 case Orientation.**LEFT**:  
 StepDown();  
 curOrientation = Orientation.**RIGHT**;  
 break;  
 case Orientation.**DOWN**:  
 StepRight();  
 curOrientation = Orientation.**UP**;  
 break;  
 case Orientation.**UP**:  
 StepRight();  
 curOrientation = Orientation.**DOWN**;  
 break;  
 default:  
 throw new ArgumentOutOfRangeException();  
 }  
   
 PaintSnake(graph);  
 }  
  
 private void StepLeft()  
 {  
 headCoordinates.X -= **SizeSegment**;  
 }  
  
 private void StepRight()  
 {  
 headCoordinates.X += **SizeSegment**;  
 }  
  
 private void StepUp()  
 {  
 headCoordinates.Y -= **SizeSegment**;  
 }  
  
 private void StepDown()  
 {  
 headCoordinates.Y += **SizeSegment**;  
 }   
   
 */// <summary>  
 /// Отрисовка змейки  
 /// </summary>  
 /// <param name="graph"></param>* private void PaintSnake(Graphics graph)  
 {  
 snake.Dequeue();  
   
 var path = new GraphicsPath();  
 var head = new RectangleF(headCoordinates, new SizeF(**SizeSegment**, **SizeSegment**));  
 path.AddRectangle(head);  
 snake.Enqueue(path);  
 foreach (var segment in snake)  
 {  
 graph.DrawPath(MyPen, segment);  
 }  
 }  
 }  
}

**Описание**

Змейка хранится в виде массива точек. Описан метод движения змейки, а также метод изменения её направления в зависимости от столкновения с препятствием или границей окна. Если препятствие ещё не было пройдено, то змейка “соединяется” с ним, запоминается точка соединения, и змейка делает полный обход препятствия, пока не дойдёт до точки соединения. Далее змейка отсоединяется от препятствия и ищет следующее.

**Внешний вид**



**Трассировка представлена в виде комментариев в коде**

**Бильярд**

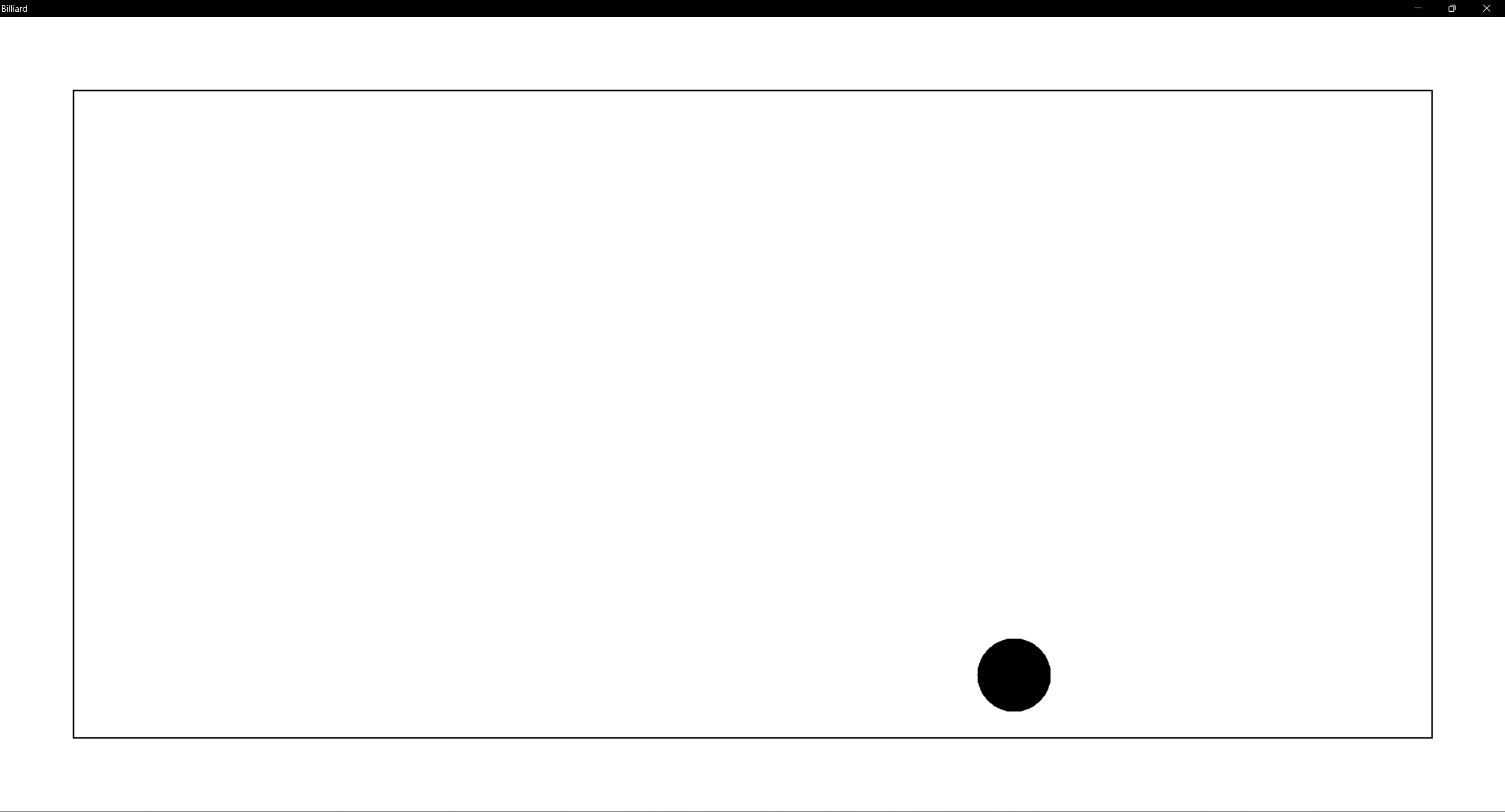
**Программная реализация**

using System;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Windows.Forms;  
  
namespace billiard  
{  
 public partial class MainForm : Form  
 {  
 private Graphics Graph;  
 private Pen MyPen;  
 private SolidBrush MyBrush;  
 private SolidBrush WhiteBrush;  
 private Timer timer;  
   
 private GraphicsPath rectangle; *// Прямоугольник/доска* private PointF[] points; *// Буффер для создания прямоугольника/доски* private int xPoint, yPoint; *// Координаты шара* private bool toRight = true, toDown = true; *// Флаги направления движения шара* private const int **BallRadius** = 50; *// Радиус шара* private const int **BallStep** = 5; *// Шаг движения шара* private const int **Fps** = 144; *// Количество кадров в секунду* public MainForm()  
 {  
 InitializeComponent();  
 Graph = CreateGraphics();  
 Graph.SmoothingMode = SmoothingMode.**HighQuality**;  
 MyPen = new Pen(Color.Black, 2);  
 MyBrush = new SolidBrush(Color.Black);  
  
 CreateBoard();  
 InitTimer(1000 / **Fps**);  
  
 xPoint = ClientSize.Width / 2;  
 yPoint = ClientSize.Height / 2;  
 }  
  
 private void InitTimer(int interval)  
 {  
 timer = new Timer();  
 timer.Interval = interval;  
 timer.Tick += timer\_tick;  
 timer.Enabled = true;  
 }  
  
 private void CreateBoard()  
 {  
 points = new PointF[4];  
 points[0] = new PointF(100, 100);  
 points[1] = new PointF(ClientSize.Width - 100, 100);  
 points[2] = new PointF(ClientSize.Width - 100, ClientSize.Height - 100);  
 points[3] = new PointF(100, ClientSize.Height - 100);  
 rectangle = new GraphicsPath();  
 rectangle.AddPolygon(points);  
 }  
  
 */// <summary>  
 /// Каждый тик перерисовывает окно  
 /// </summary>  
 /// <param name="sender"></param>  
 /// <param name="e"></param>* private void timer\_tick(object sender, EventArgs e)  
 {  
 Invalidate();  
 }  
   
 private void MainForm\_Paint(object sender, PaintEventArgs e)  
 {  
 MoveBall();  
 e.Graphics.DrawPolygon(MyPen, points);   
 e.Graphics.FillEllipse(MyBrush, xPoint, yPoint, 2 \* **BallRadius**, 2 \* **BallRadius**);  
 }  
  
 */// <summary>  
 /// Считает новые координаты шара  
 /// </summary>* private void MoveBall()  
 {  
 if (rectangle == null) return;  
  
 *// Если можем двигаться направо, то движемся направо, иначе - налево* if (toRight) xPoint += **BallStep**;  
 else xPoint -= **BallStep**;  
  
 *// Если можем двигаться вниз, то движемся вниз, иначе - вверх* if (toDown) yPoint += **BallStep**;  
 else yPoint -= **BallStep**;  
  
 *// В зависимости от пересечения прямоугольника на следующем шаге меняем направление шара* if (!rectangle.IsVisible(new Point(xPoint, yPoint + **BallStep** + 2 \* **BallRadius**))) toDown = false;  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint - **BallStep**))) toDown = true;  
 if (!rectangle.IsVisible(new Point(xPoint - **BallStep**, yPoint))) toRight = true;  
 if (!rectangle.IsVisible(new Point(xPoint + **BallStep** + 2 \* **BallRadius**, yPoint))) toRight = false;  
   
 MakeSound();  
 }  
  
 private static void MakeSound()  
 {  
 System.Media.SystemSounds.Exclamation.Play();  
 }  
  
 private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  
 {  
 Graph.Dispose();  
 MyPen.Dispose();  
 MyBrush.Dispose();  
 timer.Dispose();  
 }  
 }  
}

**Описание**

Описан метод движения шара. Если следующий шаг точки выходит за границы прямоугольника, то точка меняет направление, благодаря двум флагам, разрешающим/запрещающим двигаться в определенные стороны.

**Внешний вид**



**Трассировка представлена в виде комментариев в коде**

**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**

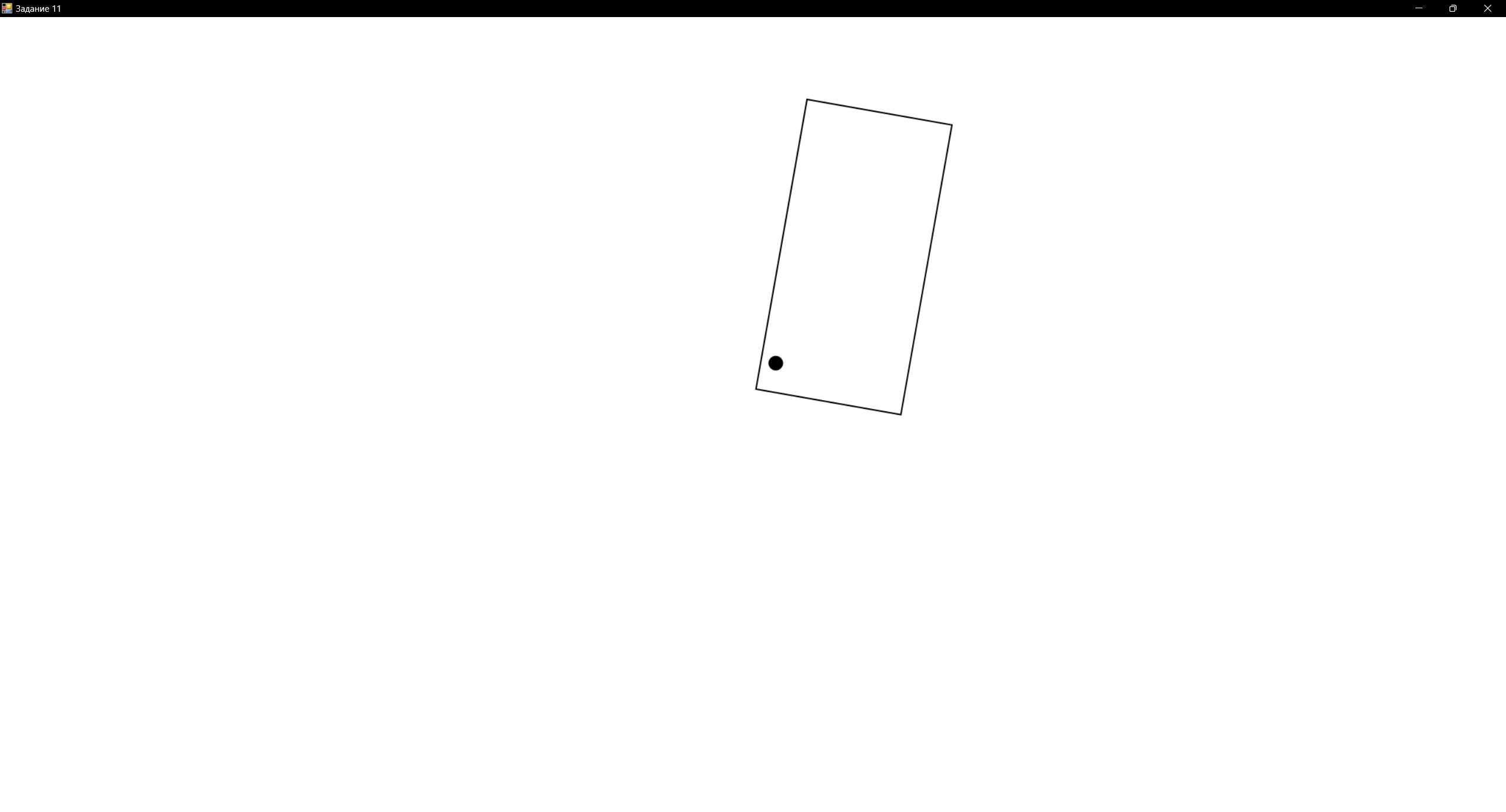
using System;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Windows.Forms;  
  
namespace task11  
{  
 public partial class MainForm : Form  
 {  
 private readonly Graphics Graph;  
 private readonly Pen MyPen;  
 private readonly SolidBrush MyBrush;  
 private Timer timer;  
   
 private GraphicsPath rectangle; *// Прямоугольник* private PointF[] points; *// Буффер для создания прямоугольника* private int xPoint, yPoint; *// Координаты точки* private bool pToRight = true, pToDown = true; *// Флаги направления движения точки* private int countRotates; *// Количество вращений прямоугольника* private bool toLeft = true; *// Направление движения прямоугольника* private const int **PRadius** = 10; *// Радиус точки* private const int **PStep** = 8; *// Шаг движения точки* private const float **RRotateAngle** = 1; *// Угол вращения прямоугольника за один тик* private int RLength = 400, RHeight = 200; *// Размеры прямоугольника* private const int **Fps** = 144; *// Количество кадров в секунду* public MainForm()  
 {  
 InitializeComponent();  
 Graph = CreateGraphics();  
 Graph.SmoothingMode = SmoothingMode.**HighQuality**;  
 MyPen = new Pen(Color.Black, 2);  
 MyBrush = new SolidBrush(Color.Black);  
  
 RLength = ClientSize.Width / 5 + 50;  
 RHeight = ClientSize.Height / 5 + 50;  
  
 InitTimer(1000 / **Fps**);  
 CreateRectangle();  
 CreatePoint();  
 }  
  
 private void InitTimer(int interval)  
 {  
 timer = new Timer();  
 timer.Interval = interval;  
 timer.Enabled = true;  
 timer.Tick += timer\_tick;  
 }  
  
 private void CreateRectangle()  
 {  
 countRotates = 0;  
 points = new PointF[4];  
 points[0] = new PointF(ClientSize.Width / 2 - RLength / 2, ClientSize.Height / 2);  
 points[1] = new PointF(ClientSize.Width / 2 + RLength / 2, ClientSize.Height / 2);  
 points[2] = new PointF(ClientSize.Width / 2 + RLength / 2, ClientSize.Height / 2 + RHeight);  
 points[3] = new PointF(ClientSize.Width / 2 - RLength / 2, ClientSize.Height / 2 + RHeight);  
 }  
  
 private void CreatePoint()  
 {  
 xPoint = ClientSize.Width / 2;  
 yPoint = ClientSize.Height / 2 + RHeight / 2;  
 }  
   
 */// <summary>  
 /// Каждый тик перерисовывает окно  
 /// </summary>  
 /// <param name="sender"></param>  
 /// <param name="e"></param>* private void timer\_tick(object sender, EventArgs e)  
 {  
 Invalidate();  
 }  
   
 private void MainForm\_Paint(object sender, PaintEventArgs e)  
 {  
 MoveRectangle();  
 MovePoint();  
 e.Graphics.DrawPolygon(MyPen, points);  
 e.Graphics.FillEllipse(MyBrush, xPoint, yPoint, 2 \* **PRadius**, 2 \* **PRadius**);  
 }  
  
 */// <summary>  
 /// Считает новые координаты прямоугольника  
 /// </summary>* private void MoveRectangle()  
 {  
 *// Если прямоугольник сделал полный круг* if (countRotates == (int)(360 / **RRotateAngle**))  
 {  
 *// Сдвигаем прямоугольник* OffsetX();  
 MakeSound();  
 *// Обнуляем счеткик поворотов* countRotates = 0;  
 }  
   
 *// Считаем новые координаты прямоугольника с учетом поворота относительно правой верхней вершины* const double **angleRadian** = **RRotateAngle** \* 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);  
   
 *// Инкрементируем счетчик поворотов* countRotates++;  
 }  
   
 */// <summary>  
 /// Сдвигает прямоугольник относительно оси X  
 /// </summary>* private void OffsetX()  
 {  
 *// Если прямоугольник движется влево, а при слудующем шаге левая вершина выйдет за пределы экрана,  
 // то меняем напрвление* if (toLeft && points[3].X - 50 < 0)  
 {  
 toLeft = false;  
 }  
  
 *// Если прямоугольник движется вправо, а при слудующем шаге правая вершина выйдет за пределы экрана,  
 // то меняем напрвление* if (!toLeft && points[0].X + 50 > ClientSize.Width)  
 {  
 toLeft = true;  
 }  
   
 *// Считаем новые координаты прямоугольник с учетом шага* 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 MakeSound()  
 {  
 System.Media.SystemSounds.Exclamation.Play();  
 }  
  
 */// <summary>  
 /// Считает новые координаты точки  
 /// </summary>* private void MovePoint()  
 {  
 if (rectangle == null) return;  
  
 *// Если можем двигаться направо, то движемся направо, иначе - налево* if (pToRight) xPoint += **PStep**;  
 else xPoint -= **PStep**;  
  
 *// Если можем двигаться вниз, то движемся вниз, иначе - вверх* if (pToDown) yPoint += **PStep**;  
 else yPoint -= **PStep**;  
  
 *// В зависимости от пересечения прямоугольника на следующем шаге меняем направление точки* if (!rectangle.IsVisible(new Point(xPoint, yPoint + **PStep** + 2 \* **PRadius**))) pToDown = false;  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint - **PStep**))) pToDown = true;  
 if (!rectangle.IsVisible(new Point(xPoint - **PStep**, yPoint))) pToRight = true;  
 if (!rectangle.IsVisible(new Point(xPoint + **PStep** + 2 \* **PRadius**, yPoint))) pToRight = false;  
 }  
  
 private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)  
 {  
 Graph.Dispose();  
 MyPen.Dispose();  
 MyBrush.Dispose();  
 timer.Dispose();  
 }  
 }  
}

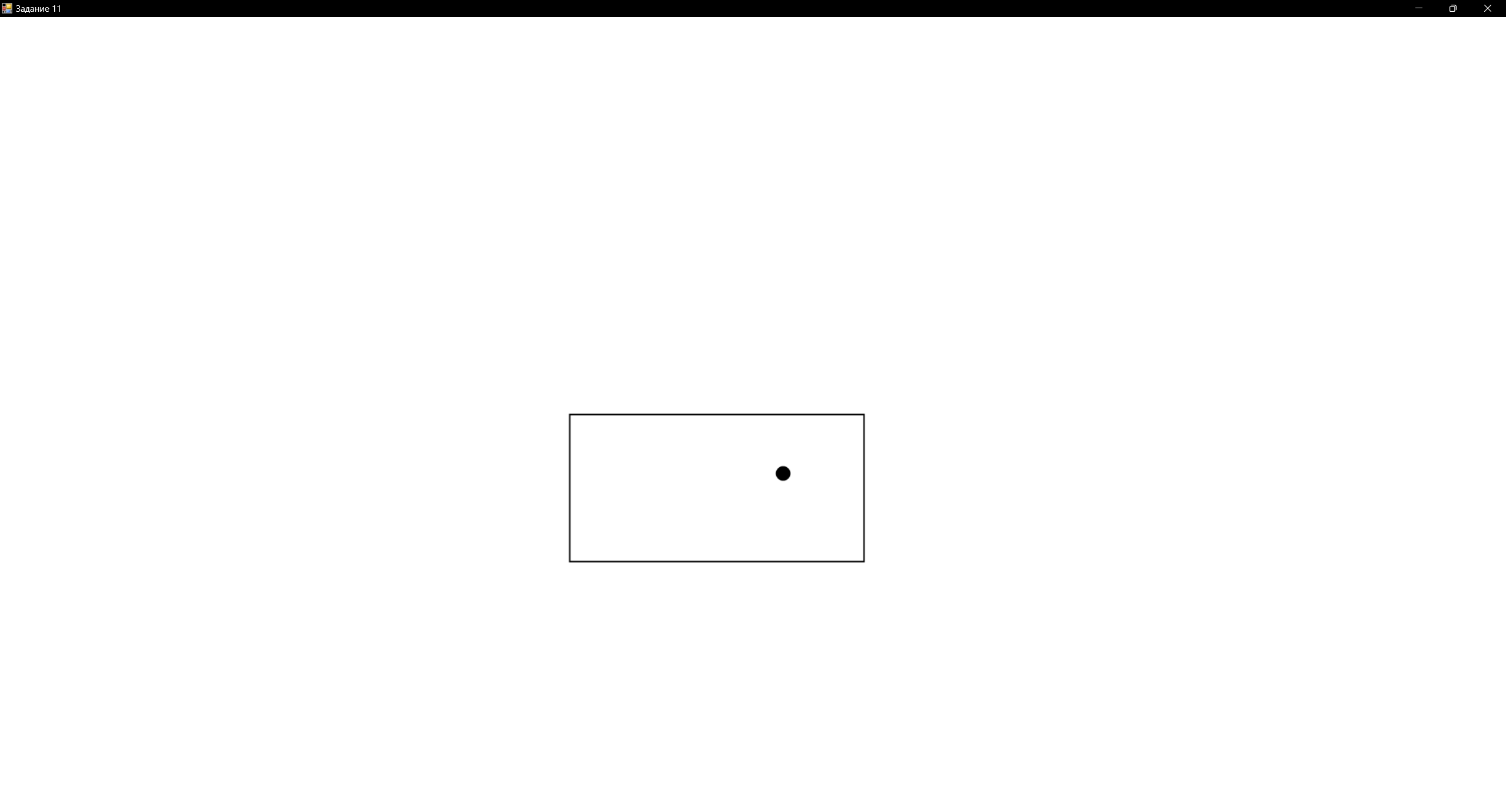
**Описание**

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**

Каждый тик таймера происходит вычисление новых координат прямоугольника и перерисовка формы.

private void MainForm\_Paint(object sender, PaintEventArgs e)  
{  
 MoveRectangle();  
 MovePoint();  
 e.Graphics.DrawPolygon(MyPen, points);  
 e.Graphics.FillEllipse(MyBrush, xPoint, yPoint, 2 \* **PRadius**, 2 \* **PRadius**);  
}

Если прямоугольник сделал полный оборот, то он смещается по оси X, и счетчик поворотов обнуляется.

*// Если прямоугольник сделал полный круг*if (countRotates == (int)(360 / **RRotateAngle**))  
{  
 *// Сдвигаем прямоугольник* OffsetX();  
 MakeSound();  
 *// Обнуляем счеткик поворотов* countRotates = 0;  
}

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()  
{  
 *// Если прямоугольник движется влево, а при слудующем шаге левая вершина выйдет за пределы экрана,  
 // то меняем напрвление* if (toLeft && points[3].X - 50 < 0)  
 {  
 toLeft = false;  
 }  
  
 *// Если прямоугольник движется вправо, а при слудующем шаге правая вершина выйдет за пределы экрана,  
 // то меняем напрвление* if (!toLeft && points[0].X + 50 > ClientSize.Width)  
 {  
 toLeft = true;  
 }  
   
 *// Считаем новые координаты прямоугольник с учетом шага* 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);  
}

Иначе считаются новые координаты фигуры с учетом поворота относительно правой верхней вершины, и инкрементируется счетчик поворотов.

*// Считаем новые координаты прямоугольника с учетом поворота относительно правой верхней вершины*const double **angleRadian** = **RRotateAngle** \* 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);  
  
*// Инкрементируем счетчик поворотов*countRotates++;

Параллельно считается новое положение точки. Если следующий шаг точки выходит за границы прямоугольника, то точка меняет направление, благодаря двум флагам, разрешающим/запрещающим двигаться в определенные стороны.

private void MovePoint()  
{  
 if (rectangle == null) return;  
  
 *// Если можем двигаться направо, то движемся направо, иначе - налево* if (pToRight) xPoint += **PStep**;  
 else xPoint -= **PStep**;  
  
 *// Если можем двигаться вниз, то движемся вниз, иначе - вверх* if (pToDown) yPoint += **PStep**;  
 else yPoint -= **PStep**;  
  
 *// В зависимости от пересечения прямоугольника на следующем шаге меняем направление точки* if (!rectangle.IsVisible(new Point(xPoint, yPoint + **PStep** + 2 \* **PRadius**))) pToDown = false;  
 if (!rectangle.IsVisible(new Point(xPoint, yPoint - **PStep**))) pToDown = true;  
 if (!rectangle.IsVisible(new Point(xPoint - **PStep**, yPoint))) pToRight = true;  
 if (!rectangle.IsVisible(new Point(xPoint + **PStep** + 2 \* **PRadius**, yPoint))) pToRight = false;  
}