**Федеральное государственное бюджетное образовательное учреждение**
**высшего образования**
**«Московский Авиационный Институт»**
**(Национальный Исследовательский Университет)**

**Факультет №8 «Компьютерные науки и прикладная математика»**
**Кафедра 805 «Прикладная математика»**

Курсовая работа
по курсу «ИСРППС»
3 семестр
Вариант 4.2

Тема:

# Раскраска спутников в зависимости от доступа к точке, отображение на сеть, построение графика уровня сигнала.

**Автор работы:**
студент 2 курса, гр. М8О-203Б-21
Барсуков Е. А.
**Руководитель проекта:**
Семенов А. С.
**Дата сдачи:**

**Москва 2022**

Содержание:

## Постановка задачи.

Целью моей курсовой работы является создание программного продукта - модели для отслеживания движения спутников на орбите Земли.

Задачи, которые требуется реализовать:

- удобный масштабируемый интерфейс,
- возможность добавлять и удалять спутники,
- график, показывающий “уровень сигнала” для выбранной точки на Земле,
- решетка, показывающая состояние системы спутников

Для реализации проекта я использовал язык C# и Windows Forms, основанные на .NET Core 3.1. Так же в разработке была использована библиотека LiveCharts для удобной работы с графиками.

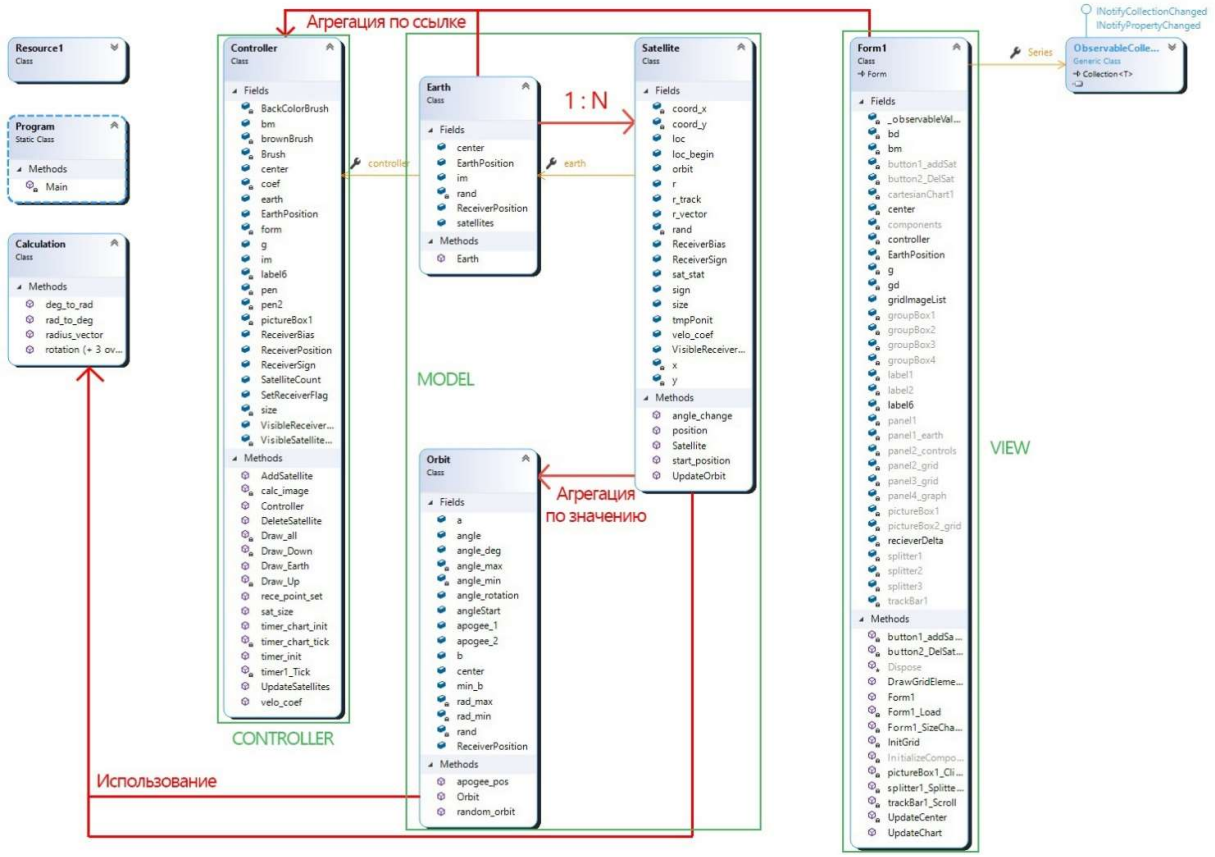
# Описание используемой технологии разработки.

Для создания программного продукта использовалась среда Windows Forms. Это интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки, т.е. разработка может идти как на, например, C#, так и на C++.

.NET (ранее известна как .NET Core) — это модульная платформа для разработки программного обеспечения с открытым исходным кодом. Используемая мной версия .NET Core 3.1 вышла 3 декабря 2019 года и является версией LTS (Long term support), т.е. версией с долгосрочной поддержкой. .NET основана на .NET Framework. Платформа .NET отличается от неё модульностью, кроссплатформенностью, возможностью применения облачных технологий.

.NET — модульная платформа. Каждый её компонент обновляется через менеджер пакетов NuGet, а значит можно обновлять её модули по отдельности, в то время как .NET Framework обновляется целиком. Каждое приложение может работать с разными модулями и не зависит от единого обновления платформы. В своем проекте я использовал NuGet для установки LiveCharts – мощного инструмента для работы с графиками.

# Архитектура приложения.




В основе программы лежит паттерн MVC. Он представлен моделью – набором классов Earth, Satellite и Orbit. Отношения между классами представлены на диаграмме выше.

Класс Earth служит контейнером для спутников и содержит информацию о координатах Земли в форме.

Класс Satellite представляет спутник на орбите. Каждый спутник содержит объект класса Orbit – орбиту, которая, в свою очередь, реализует средства для работы с орбитой.

Программа использует класс Calculation, содержащий некоторые полезные функции для, например, расчёта орбит.

В классе Resource1 хранятся графические ресурсы программы – изображение Земли и составляющие решетки спутников.

# Текст исходного кода.

Класс Earth:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Threading.Tasks;

namespace Satellite_Tracking_Proj.Model
{
    public class Earth
    {
        public Controller.Controller controller { set; get; } = null;
        Random rand = new Random();
        public List<Satellite> satellites; //list of all satellites

        public Earth(Controller.Controller controller, Point center, Point ReceiverPosition)
        {
            this.controller = controller;
            this.satellites = new List<Satellite>();

            this.center = center;
            this.ReceiverPosition = ReceiverPosition;
            this.EarthPosition = new Point(center.X - 50, center.Y - 50);
            this.im = Resources.Resource1.earth_small2_test;
        }

        public Point center;
        public Point ReceiverPosition; // position of reciever point
        public Point EarthPosition; // EarthPosition
        public Image im;
    }
}
```

Класс Satellite:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Drawing;
using System.Windows.Forms;

namespace Satellite_Tracking_Proj.Model
{
    public class Satellite
    {
        public Earth earth { set; get; }

        public Point loc;
        /*========= for intersecting orbits */
        public Point tmpPonit;
        /*=========*/
        int x, y; // start position
        double coord_x, coord_y;
        public double r; // radius-vector
        public double r_vector;
        // ***********
        public double r_track;
        public int sat_stat;
        public bool ReceiverSign = true; // true - for increasing x (+), false - .;. (-)
        public double ReceiverBias;
```

```
public bool VisibleReceiverFlag = true;
public Orbit orbit = null;
/*-------------*/
public double velo_coef; // коэфф скорости
public int size;
/*-------------*/
Random rand = new Random();
public Point loc_begin;
public int sign; // направл вращения

public Satellite(Point center, Point ReceiverPosition)
{
    this.tmpPonit = new Point();
    this.orbit = new Orbit(center, ReceiverPosition);
    loc_begin = new Point();
    start_position();
    this.orbit.center = center;
    this.orbit.ReceiverPosition = ReceiverPosition;
    loc_begin = loc;
    sign = rand.Next(0, 100);
    velo_coef = 1;
    size = 4;
}

public void UpdateOrbit(Point cent, Point npos)
{
    this.orbit.center = cent;
    this.orbit.ReceiverPosition = npos;
}

public void start_position()
{
    this.coord_x = orbit.b * Math.Cos(orbit.angle_rotation);
    this.coord_y = orbit.a * Math.Sin(orbit.angle_rotation);
    // rotating x and y
    Calculation.rotation(ref coord_x, ref coord_y, orbit.angle);

    this.x = (int)(orbit.center.X + coord_x);
    this.y = (int)(orbit.center.Y + coord_y);

    this.loc = new Point(x - size, y - size); // default location
}

public void position() // position every tick
{
    this.coord_x = orbit.b * Math.Cos(orbit.angle_rotation);
    this.coord_y = orbit.a * Math.Sin(orbit.angle_rotation);
    // rotating x and y
    Calculation.rotation(ref coord_x, ref coord_y, orbit.angle);
    this.x = (int)(orbit.center.X + coord_x);
    this.y = (int)(orbit.center.Y + coord_y);
    this.loc = new Point(x - size, y - size);

    // calc radius-vector to reciever
    r_track = Calculation.radius_vector(loc, orbit.ReceiverPosition);
    double del_x, del_y;
    del_x = loc.X - orbit.center.X;
    del_y = loc.Y - orbit.center.Y;
    r = Math.Sqrt(Math.Pow(del_x, 2) + Math.Pow(del_y, 2)); // нашли гипотенузу
}

public void angle_change(Point loc)
```

```
        {
            if (sign > 50)
            {
                orbit.angle_rotation -= 0.01 * velo_coef;
            }
            else
            {
                orbit.angle_rotation += 0.01 * velo_coef;
            }
            Task.Delay(1);

            /* sign settings for reciever position */
            if (ReceiverSign == true)
            {
                this.ReceiverBias += 0.005;
            }
            else
            {
                this.ReceiverBias -= 0.005;
            }
            //for intersecting orbits
            this.tmpPonit = loc;
        }
    }
}
```

Класс Orbit:

```
using System;
using System.Drawing;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Satellite_Tracking_Proj.Model
{
    public class Orbit
    {
        int angle_min = 0;
        int angle_max = 180;
        int rad_min = 52;
        int rad_max = 150;

        public int a = 150;
        public int b;
        public int min_b = 0;
        public double angle;
        //
        public Point center;
        public Point ReceiverPosition;
        //
        public double angle_rotation;
        public double angleStart;
        public double angle_deg;

        /*=========*/
        public Point apogee_1; //верхний(левый) и
        public Point apogee_2; //нижний(правый)
        /*=========*/

        Random rand = new Random();
        public Orbit(Point center, Point ReceiverPosition)
```

```
        {
            random_orbit();
            this.angle_deg = angle;
            Calculation.rad_to_deg(ref angle_deg);
        }

        public void random_orbit()
        {
            //задаем слачайные радиусы эллипсов и углы
            this.b = rand.Next(rad_min, rad_max);
            this.angle = rand.Next(angle_min, angle_max); //
            this.angle = this.angle * Math.PI / 180.0; // 0 .. pi
        }
    }
}
```

Класс Controller:

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

namespace Satellite_Tracking_Proj.Controller
{
    public class Controller
    {
        Form1 form = null;
        public Model.Earth earth = null;

        public Point center;
        public Point EarthPosition;

        /*=========*/
        public int SetReceiverFlag; // flag for mouse set position
        public Point ReceiverPosition; // point of rece -> to Model
        public bool ReceiverSign = true; // true - for increasing x (+), false - .;. (-)
        public double ReceiverBias; // bias of receiver
        public bool VisibleReceiverFlag = true; // -> controller */ // flag for rece dot
visibility
        /*=========*/

        private int coef; // for every sat
        private int VisibleSatelliteCount; // visible satellite count
        private int size; // size of satellite
        public int SatelliteCount;

        // Create solid brush.
        SolidBrush Brush = new SolidBrush(Color.BlueViolet);
        SolidBrush brownBrush = new SolidBrush(Color.Brown);

        Pen pen = new Pen(Color.Black);
        Pen pen2 = new Pen(Color.Red);

        public Bitmap bm;
        public Graphics g;
        public Image im = Resources.Resource1.earth_small2_test; // of Earth (path)
        PictureBox pictureBox1;

        /*=========*/
```

```
        SolidBrush BackColorBrush; // for invisibility
        /*=========*/

        //--for debug
        Label label6;
        // +++++++++

        public Controller(
            Form1 form,
            ref Graphics g,
            ref Bitmap bm,
            ref Point center,
            ref PictureBox pictureBox1,
            ref Point EarthPosition,
            ref Label label6
            )
        {
            MessageBox.Show("Выберите точку на поверхности Земли", "(!)",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            this.form = form;

            this.g = g;
            this.bm = bm;
            this.center = center;
            this.pictureBox1 = pictureBox1;
            this.EarthPosition = EarthPosition;

            this.coef = 1; // коэффициент скорости, который домножается на угол смещения
            this.size = 4; // размер спутника
            this.SetReceiverFlag = 0; // флаг, отвечающий за существование точки приемника
            this.label6 = label6;

            /*=========*/
            Color brushColor = Color.FromArgb(250 / 100 * 25, 255, 0, 0); // invisible color
            this.BackColorBrush = new SolidBrush(brushColor);
            /*=========*/

            earth = new Model.Earth(this, center, ReceiverPosition);
            this.EarthPosition = earth.EarthPosition;
            this.im = earth.im;
            Draw_Earth(this.im, this.EarthPosition);
            pictureBox1.Image = bm;


        }

        public void timer_init(object sender, EventArgs e)
        {
            Timer x = new Timer();
            x.Interval = 1; // 1ms
            x.Start();
            x.Tick += new EventHandler(timer1_Tick);
        }

        public void timer_chart_init(object sender, EventArgs e)
        {
            Timer t = new Timer();
            t.Interval = 200; // 200ms
            t.Start();
            t.Tick += new EventHandler(timer_chart_tick);
        }
        private void timer_chart_tick(object sender, EventArgs e)
```

```
        {
            form.UpdateChart(VisibleSatelliteCount);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {

            g.Clear(pictureBox1.BackColor);
            GC.Collect();
            Draw_Earth(this.im, this.EarthPosition);


            // changing ReceiverPosition
            Point tmp_rece_point = new Point(ReceiverPosition.X - 3, ReceiverPosition.Y - 3);
            Rectangle rec = new Rectangle(tmp_rece_point, new Size(6, 6));
            // Fill ellipse on screen.
            g.FillEllipse(brownBrush, rec);

            int counter = 0;
            this.VisibleSatelliteCount = 0;
            foreach (var satellite in this.earth.satellites)
            {
                satellite.position();
                satellite.velo_coef = coef;
                String tmpstr = Convert.ToString(satellite.orbit.b);

                int visi_flag = 0; //

                if (satellite.r_track <= satellite.orbit.b) // рад-вектор до сата из ресивера
                {
                    visi_flag += 1;
                    this.VisibleSatelliteCount += 1;
                    Point tmp = new Point(satellite.loc.X + size, satellite.loc.Y + size);
                    g.DrawLine(pen2, ReceiverPosition, tmp);

                    //update satellite icon
                    form.DrawGridElement(((counter) % 9) * 100, ((counter) / 9) * 100,
Resources.Resource1.sat_0);
                }
                else
                {
                    form.DrawGridElement(((counter) % 9) * 100, ((counter) / 9) * 100,
Resources.Resource1.sat_1);
                }

                if (visi_flag == 1)
                {
                    this.Brush = new SolidBrush(Color.Red);
                }
                else
                {
                    this.Brush = new SolidBrush(Color.BlueViolet);
                }
                Draw_all(satellite.loc, satellite.orbit.a, satellite.orbit.b,
satellite.orbit.angleStart, satellite.orbit.angle_deg, this.Brush);

                pictureBox1.Image = bm;
                satellite.angle_change(satellite.loc);
                counter++;
            }
            pictureBox1.Image = bm;
            //label6.Text = "Visible Satellites: " + this.VisibleSatelliteCount;
```

```
        }

        public DialogResult AddSatellite()
        {
            //добавляем в список новый объект
            if (SatelliteCount == 81)
            {
                MessageBox.Show("Нельзя добавить больше 81 спутника!");
                return DialogResult.Abort;
            }
            else
            {
                if (SetReceiverFlag == 0) // enum type ++
                {
                    return DialogResult.Retry;
                }
                else
                {
                    earth.satellites.Add(new Model.Satellite(center, ReceiverPosition));  //
Satellite ++
                    this.SatelliteCount = earth.satellites.Count;
                    if (SatelliteCount == 1)
                    {
                        form.DrawGridElement(0 * 100, (SatelliteCount / 9) * 100,
Resources.Resource1.sat_1);
                    }
                    else
                    {
                        form.DrawGridElement(((SatelliteCount - 1) % 9) * 100,
((SatelliteCount - 1) / 9) * 100, Resources.Resource1.sat_1);
                    }

                    return DialogResult.OK;
                }
            }
        }

        public void UpdateSatellites(Point RecPoint)
        {
            foreach (var sat in earth.satellites)
            {
                sat.UpdateOrbit(center, RecPoint);
            }
        }

        public DialogResult DeleteSatellite()
        {
            if (earth.satellites.Count == 0)
            {
                return DialogResult.No;
            }
            else
            {
                earth.satellites.RemoveAt(earth.satellites.Count - 1);
                this.SatelliteCount = earth.satellites.Count;
                form.DrawGridElement(((SatelliteCount) % 9) * 100, ((SatelliteCount) / 9) *
100, calc_image((SatelliteCount) % 9, (SatelliteCount) / 9));
                return DialogResult.OK;
            }
        }
```

```
private Image calc_image(int i, int j)
{
    if (i == 0)
    {
        //first lines
        if (j == 0)
        {
            return Resources.Resource1.grid1;

        }
        else if (j == 9- 1)
        {
            return Resources.Resource1.grid7;

        }
        else
        {
            return Resources.Resource1.grid5;

        }
    }
    else if (i != 0 && i != 9 - 1)
    {
        if (j == 0)
        {
            return Resources.Resource1.grid2;
        }
        else if (j > 0 && j != 9 - 1)
        {
            return Resources.Resource1.grid4;
        }
        else
        {
            return Resources.Resource1.grid8;
        }
    }
    else
    {
        if (j == 0)
        {
            return Resources.Resource1.grid3;
        }
        else if (j > 0 && j != 9 - 1)
        {
            return Resources.Resource1.grid6;
        }
        else
        {
            return Resources.Resource1.grid9;
        }
    }
}

public void velo_coef(int coef)
{
    this.coef = coef;
}

public void sat_size(int size)
{
    this.size = size;
}
```

```
        public DialogResult rece_point_set(Point point, double rad)// enum type !!!!!!!!!
        {

            int r = (int)Math.Sqrt(Math.Pow(center.X - point.X, 2) + Math.Pow(center.Y -
point.Y, 2));
            if (SetReceiverFlag == 0) // nothin set
            {
                if (r <= rad)
                {

                    this.ReceiverPosition = point; //
                    this.SetReceiverFlag = 1;
                    return DialogResult.OK;
                }
                else
                {
                    return DialogResult.Retry;
                }
            }
            else
            {
                return DialogResult.Cancel;
            }
        }

        private void Draw_Up(Point loc, int a, int b, double angleStart, double angle,
SolidBrush brush)
        {
            Rectangle sat = new Rectangle(loc, new Size(size * 2, size * 2)); //
            // Fill ellipse on screen.
            g.FillEllipse(brush, sat);
            //g.DrawImage(im, EarthPosition);
            g.TranslateTransform(center.X, center.Y); //это центр вращения
            g.RotateTransform((float)angle); //Поворачиваем
            g.TranslateTransform(-center.X, -center.Y);
            g.DrawArc(pen, (float)center.X - b, (float)center.Y - a, b * 2, a * 2,
(float)angleStart, 360 - (float)angleStart * 2);
            g.ResetTransform(); //Возвращаем точку отчета на 0, чтоб дальше рисовать как
обычно
        }

        private void Draw_Down(Point loc, int a, int b, double angleStart, double angle,
SolidBrush brush)
        {
            //g.DrawImage(im, EarthPosition);
            Rectangle sat = new Rectangle(loc, new Size(size * 2, size * 2));
            // Fill ellipse on screen.
            g.TranslateTransform(center.X, center.Y); //это центр вращения
            g.RotateTransform((float)angle); //Поворачиваем
            g.TranslateTransform(-center.X, -center.Y);
            g.DrawArc(pen, (float)center.X - b, (float)center.Y - a, b * 2, a * 2,
(float)angleStart, 360 - (float)angleStart * 2);
            g.ResetTransform(); //Возвращаем точку отчета на 0, чтоб дальше рисовать как
обычно
            g.FillEllipse(brush, sat);
        }

        private void Draw_all(Point loc, int a, int b, double angleStart, double angle,
SolidBrush brush)
        {
            Draw_Up(loc, a, b, angleStart, angle, brush);
```

```
            Draw_Down(loc, a, b, angleStart, angle, brush);
        }

        public void Draw_Earth(Image im, Point EarthPosition)
        {
            g.DrawImage(im, EarthPosition.X, EarthPosition.Y,
Resources.Resource1.earth_small2_test.Width/6,
Resources.Resource1.earth_small2_test.Height/6);
        }


    }
}
```

Класс Form1:

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

namespace Satellite_Tracking_Proj
{
    public partial class Form1 : Form
    {
        private Point center;
        private Point EarthPosition;
        private Point recieverDelta;


        Controller.Controller controller = null;
        public List<Image> gridImageList = new List<Image>();
        Graphics g;
        Graphics gd;
        Bitmap bm;
        Bitmap bd;
        //debug
        Label label6;

        public Form1()
        {
            InitializeComponent();
            bm = new Bitmap(pictureBox1.Width, pictureBox1.Height); // bitmap for drawing
earth model
            _observableValues = new ObservableCollection<ObservableValue> // graph setup
            {
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
```

```
                new ObservableValue(0),
                new ObservableValue(0),
                new ObservableValue(0),
            };

            Series = new ObservableCollection<ISeries>
            {
                new LineSeries<ObservableValue>
                {
                    Values = _observableValues,
                    Fill = null,
                    GeometrySize = 0,
                    LineSmoothness = 0.1,
                    AnimationsSpeed = TimeSpan.FromMilliseconds(1)
                }
            };

            g = Graphics.FromImage(bm);
            center = new Point(pictureBox1.Width / 2, pictureBox1.Height / 2); // center of
the earth
            EarthPosition = new Point(center.X - 50, center.Y - 50);
            pictureBox1.Image = bm;
            controller = new Controller.Controller(
                this,
                ref g,
                ref bm,
                ref center,
                ref pictureBox1,
                ref EarthPosition,
                ref label6
                );

            var xAxis = new Axis // graph limits
            {
                MaxLimit = 10,
                MinLimit = 0,
            };
            var yAxis = new Axis
            {
                MinLimit = -0.2
            };
            cartesianChart1.Series = Series;
            cartesianChart1.XAxes = new List<Axis> { xAxis };
            cartesianChart1.YAxes = new List<Axis> { yAxis };

            pictureBox2_grid.SizeMode = PictureBoxSizeMode.Zoom;
            InitGrid(9); // creating initial grid 9x9
        }

        private ObservableCollection<ObservableValue> _observableValues;
        public ObservableCollection<ISeries> Series { get; set; }

        public void UpdateChart(int signalStrength) // graph updater, fires every 200ms. adds
new point and deletes outdated one.
        {
            _observableValues.Add(new ObservableValue(signalStrength));
            _observableValues.RemoveAt(0);
        }

        public void DrawGridElement(int x, int y, Image image) // used to update grid
elements. draws spicified IMAGE at specified location
        {
```

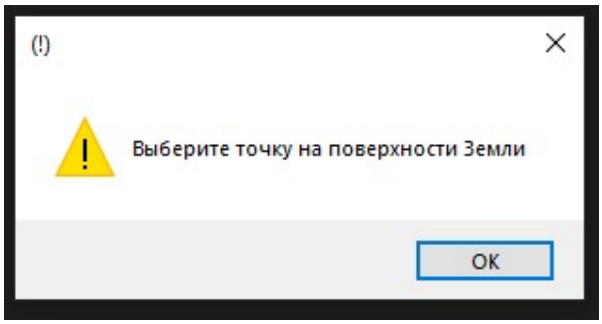
```
            gd.DrawImage(image, x, y, 100, 100);
            pictureBox2_grid.Image = bd;
        }
        private void Form1_Load(object sender, EventArgs e) // start timers
        {
            controller.timer_init(sender, e);
            controller.timer_chart_init(sender, e);
        }

        private void UpdateCenter() // updates center and earth position variables, fires
every time form size changes.
        {
            bm = new Bitmap(pictureBox1.Width, pictureBox1.Height);
            g = Graphics.FromImage(bm);
            controller.bm = bm;
            controller.g = g;
            controller.center = new Point(pictureBox1.Width / 2, pictureBox1.Height / 2);
            center = controller.center;
            if (controller.SetReceiverFlag == 1)
            {
                controller.ReceiverPosition.X += (center.X - controller.ReceiverPosition.X) -
recieverDelta.X;
                controller.ReceiverPosition.Y += (center.Y - controller.ReceiverPosition.Y) -
recieverDelta.Y;
                controller.UpdateSatellites(controller.ReceiverPosition);
            }

            EarthPosition = new Point(center.X - 50, center.Y - 50);
            controller.EarthPosition = EarthPosition;
        }

        private void Form1_SizeChanged(object sender, EventArgs e)
        {
            UpdateCenter();
        }

        private void button1_addSat_Click(object sender, EventArgs e) // add new satellite
        {
            DialogResult dialogResult;
            dialogResult = controller.AddSatellite();
            switch (dialogResult)
            {
                case DialogResult.OK:
                    break;
                case DialogResult.Retry:
                    MessageBox.Show(" Для начала задайте точку приемника \n Тыкните в любое
место на Земле", "(!)", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    break;
            }
        }

        private void pictureBox1_Click(object sender, EventArgs e) // used to define reciever
position at startup
        {
            var mouseEventArgs = e as MouseEventArgs;
            Point point = mouseEventArgs.Location;
            DialogResult dialogResult;
            dialogResult = controller.rece_point_set(point, bm.Width/6-15);

            recieverDelta.X = center.X - point.X;
            recieverDelta.Y = center.Y - point.Y;
```

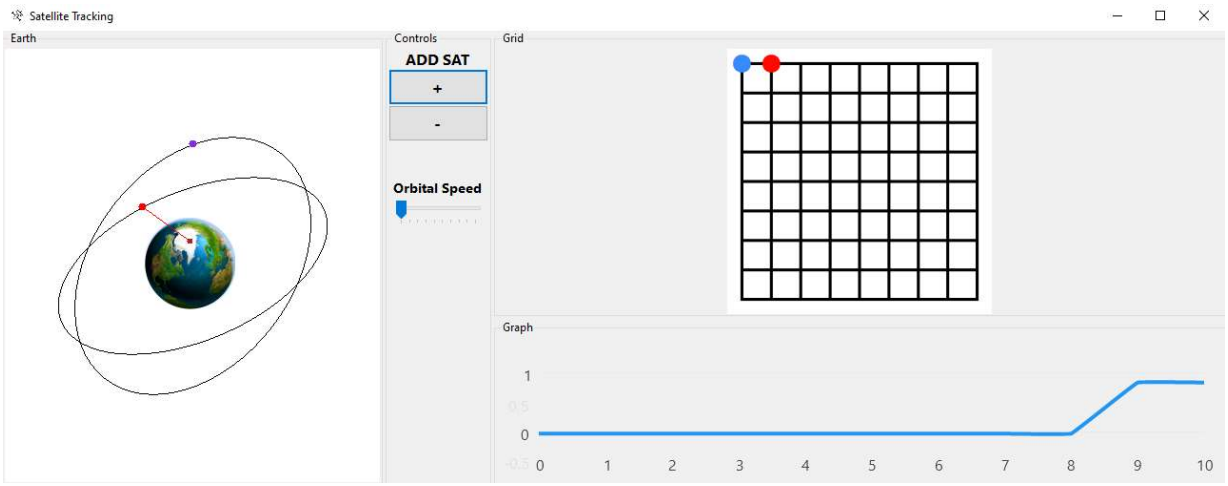
```
            switch (dialogResult)
            {
                case DialogResult.Retry:
                    MessageBox.Show("Точка должна быть 'внутри' Земли ", "(!)",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                    break;
                case DialogResult.OK:
                    MessageBox.Show("Точка задана успешно !", "DONE", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                    break;
                case DialogResult.Cancel:
                    MessageBox.Show("Точка уже была задана до этого ", "(!)",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    break;
            }
        }

        private void splitter1_SplitterMoved(object sender, SplitterEventArgs e) // if earth
picturebox size was changed
        {
            UpdateCenter();
        }
        private void InitGrid(int size) // creates grid grapic
        {
            Image imag = Resources.Resource1.grid1;
            bd = new Bitmap(size*100, size*100);
            gd = Graphics.FromImage(bd);
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    if (i == 0)
                    {
                        //first lines
                        if (j == 0)
                        {
                            gd.DrawImage(Resources.Resource1.grid1, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid1);

                        } else if (j == size - 1)
                        {
                            gd.DrawImage(Resources.Resource1.grid7, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid7);

                        } else
                        {
                            gd.DrawImage(Resources.Resource1.grid5, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid5);

                        }
                    } else if (i != 0 && i != size-1)
                    {
                        if (j == 0)
                        {
                            gd.DrawImage(Resources.Resource1.grid2, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid2);
                        } else if (j > 0 && j != size - 1)
                        {
```

```
                            gd.DrawImage(Resources.Resource1.grid4, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid4);
                        } else
                        {
                            gd.DrawImage(Resources.Resource1.grid8, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid8);
                        }
                    } else
                    {
                        if (j == 0)
                        {
                            gd.DrawImage(Resources.Resource1.grid3, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid3);
                        } else if (j > 0 && j != size - 1)
                        {
                            gd.DrawImage(Resources.Resource1.grid6, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid6);
                        } else
                        {
                            gd.DrawImage(Resources.Resource1.grid9, i * 100, j * 100, 100,
100);
                            gridImageList.Add(Resources.Resource1.grid9);
                        }
                    }
                }
            }
            pictureBox2_grid.Image = bd;
        }

        private void button2_DelSat_Click(object sender, EventArgs e) // delete satellite
        {
            DialogResult dialogResult;
            dialogResult = controller.DeleteSatellite();

            switch (dialogResult)
            {
                case DialogResult.OK:
                    break;
                case DialogResult.No:
                    MessageBox.Show(" Нет объектов в памяти! ", "(!)", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                    break;
            }
        }

        private void trackBar1_Scroll(object sender, EventArgs e) // update sat speed
        {
            controller.velo_coef(trackBar1.Value);
        }
    }
}
```

# Интерфейс.

При старте пользователю предлагается выбрать точку на поверхности Земли, которая будет являться приемником сигналов со спутников.



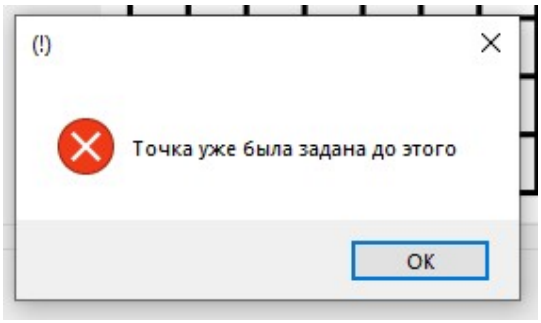



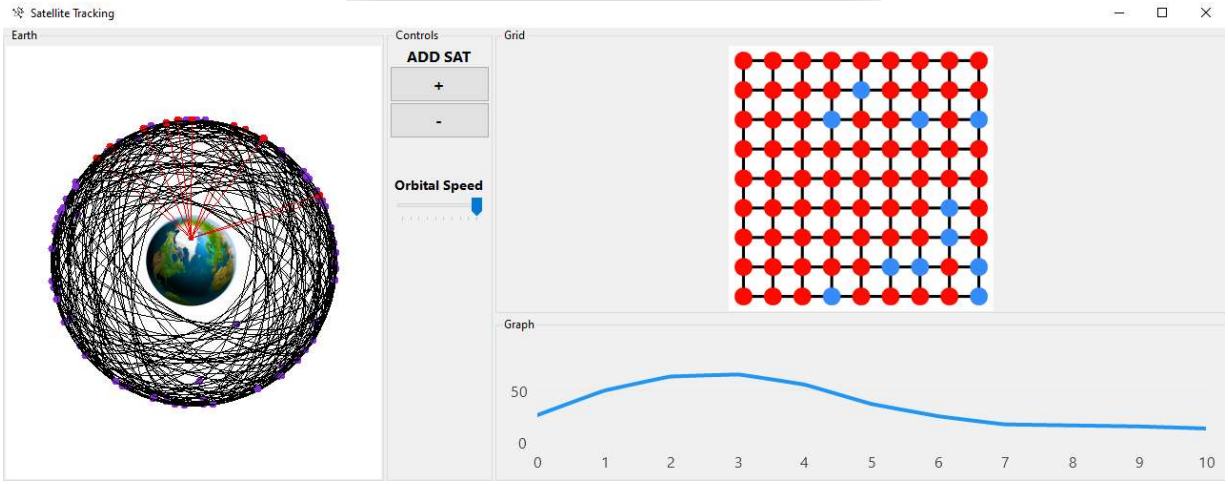

Основной интерфейс программы состоит из 4 панелей, размер которых можно произвольно изменять (интерфейс адаптируемый). На панели Earth изображена Земля и вращающиеся добавленные спутники. Панель Controls позволяет добавлять новые спутники, удалять уже существующие и менять скорость спутников.

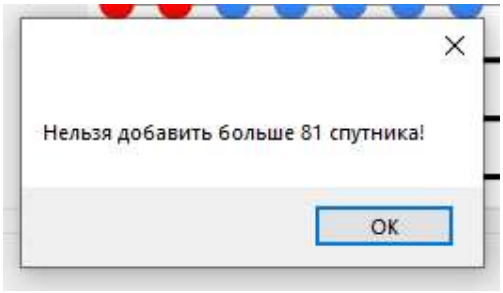
Панель Grid служит для отображения существующих спутников на решетку. Максимальное количество спутников в программе – 81. Синим обозначаются спутники, которые в данный момент подключены к передатчику на Земле, а красным – находящиеся вне зоне действия сигнала.

Панель Graph содержит график уровня сигнала. Он зависит от количества подключенных к приемнику спутников и обновляется каждые 200 мс. По горизонтальной оси отмечено время (1 деление = 200 мс), а по вертикальной – количество подключенных спутников.

В программе реализована обработка ошибок:

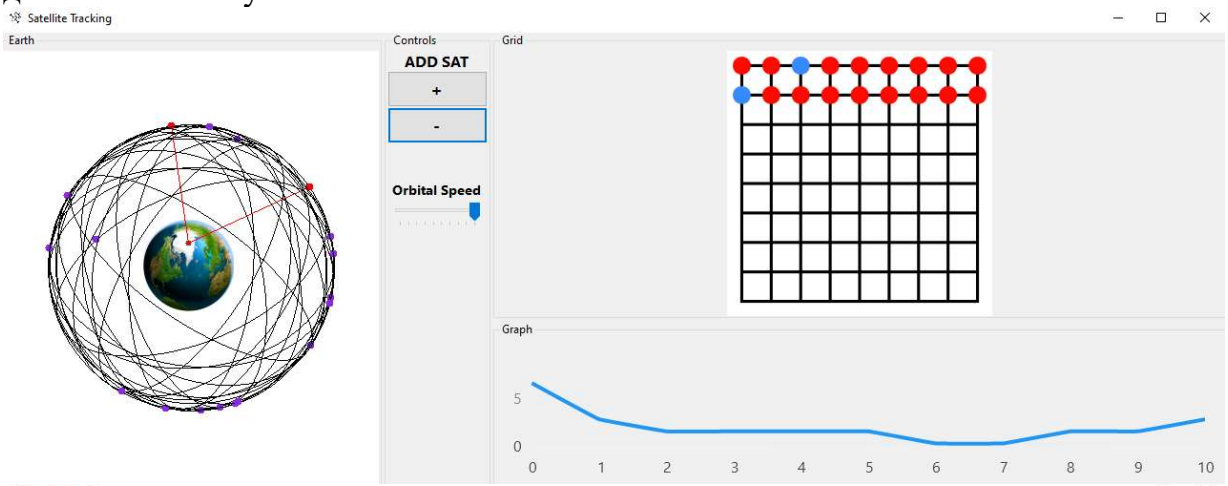

(!)
Точка уже была задана до этого
OK

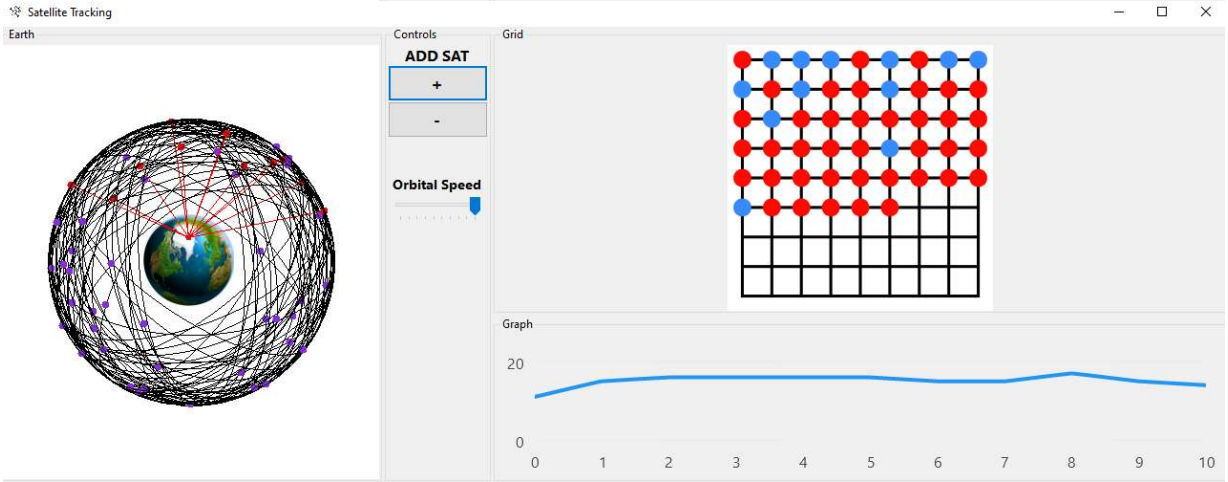

Satellite Tracking
Earth
Controls
ADD SAT
+
-
Orbital Speed
Grid
Graph
50
0
0 1 2 3 4 5 6 7 8 9 10


Нельзя добавить больше 81 спутника!
OK

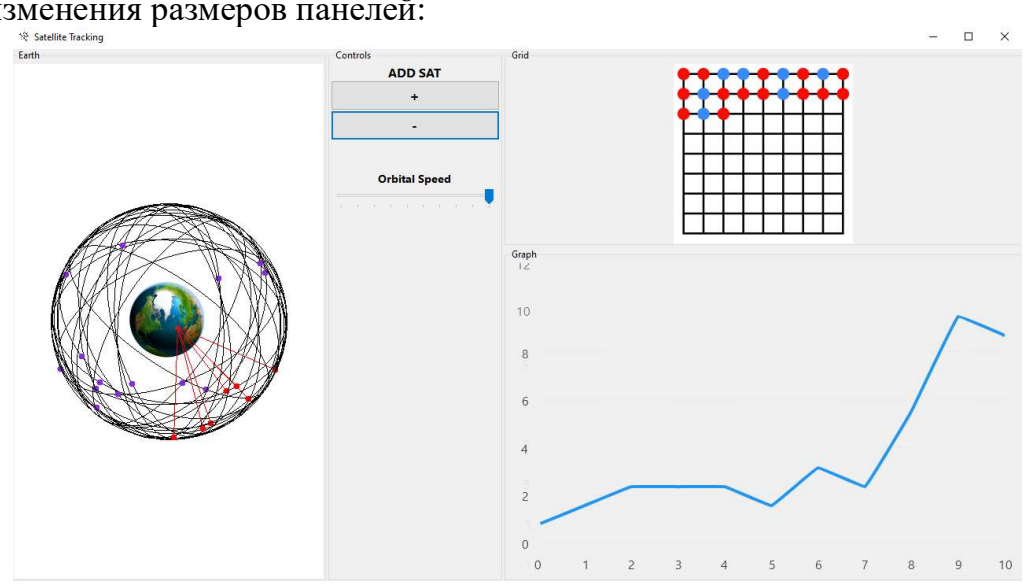
# Тесты.

Приводятся несколько скриншотов работы при разном количестве добавленных спутников:



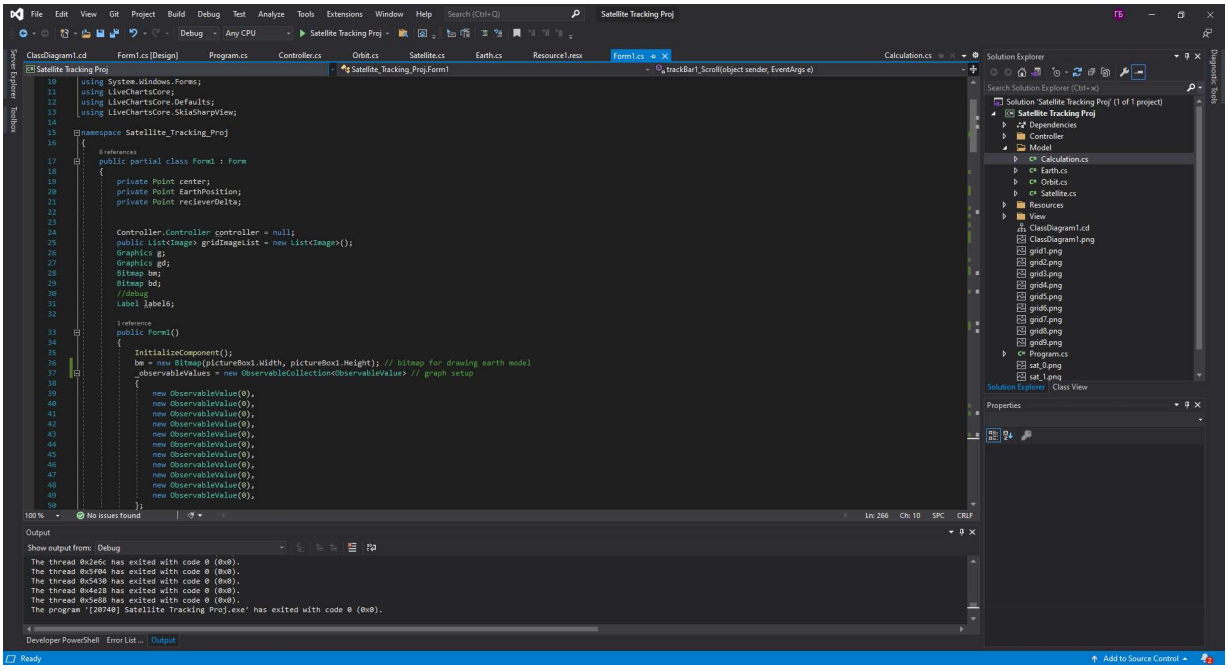




Тест изменения размеров панелей:



# Программное обеспечение.

Программный продукт разрабатывался в среде Microsoft Visual Studio 2017 - интегрированной среде разработки программного обеспечения и ряда других инструментов. Visual Studio позволяет разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms (как в случае данного проекта), UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight.



Так же была использована библиотека визуализации данных LiveCharts, которая имеет широкий набор инструментов для визуализации графиков. Для установки был использован описанный выше NuGet.

# Вывод.

В ходе работы над проектом мною было разработано приложение для отслеживания движения спутников вокруг Земли и отображения уровня сигнала на график и решетку. Был применен паттерн MVC и использовано подключение сторонних библиотек. Было показано применение знаний, полученных за курс: реализованы различные отношения между классами, применены техники объектно-ориентированного программирования.

Итоговый программный продукт выполняет поставленную задачу.

# Список использованной литературы и материалов:


1. Шарп Джон. Microsoft Visual C#. Подробное руководство. 8-е изд. — СПб.: Питер, 2017. — 848 с.
2. https://lvcharts.com/docs/winforms/2.0.0-beta.700/gallery
3. https://learn.microsoft.com/ru-Ru/previous-versions/visualstudio/visual-studio-2017/ide/whats-new-visual-studio-2017?view=vs-2017&viewFallbackFrom=vs-2019