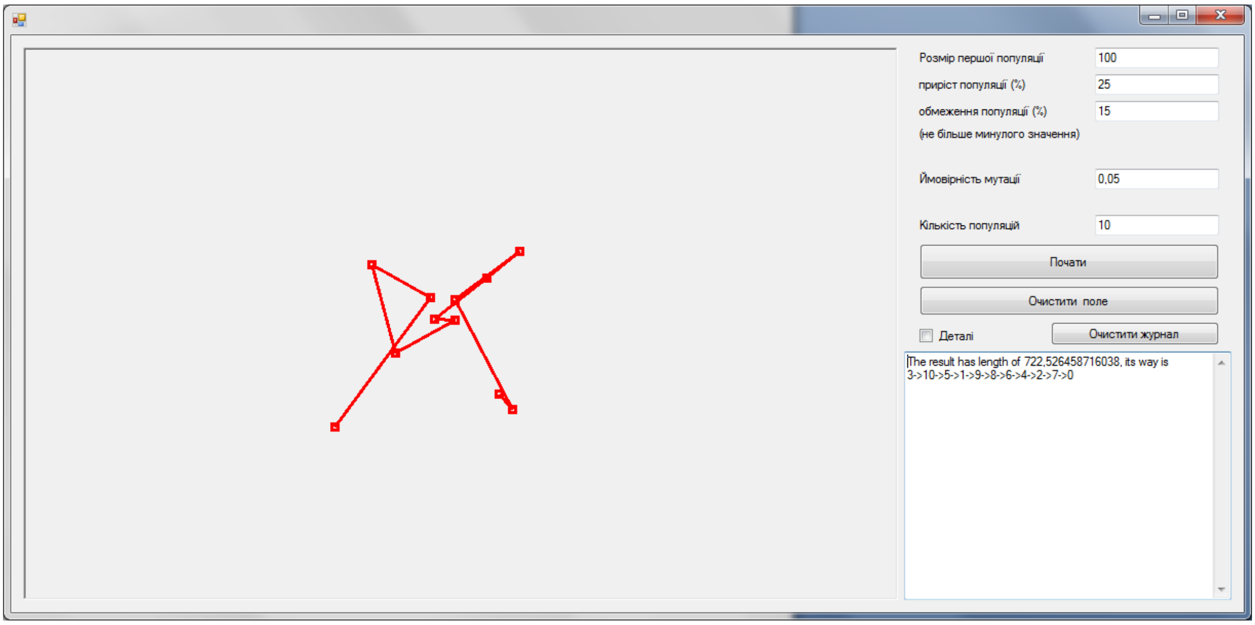
Програма вирішує задачу комівояжера. Спочатку користувач повинен розташувати точки на полі, які потім з’єднає. Також користувач може вибрати різні налаштування роботи генетичного алгоритму



Розмір першої популяції 100

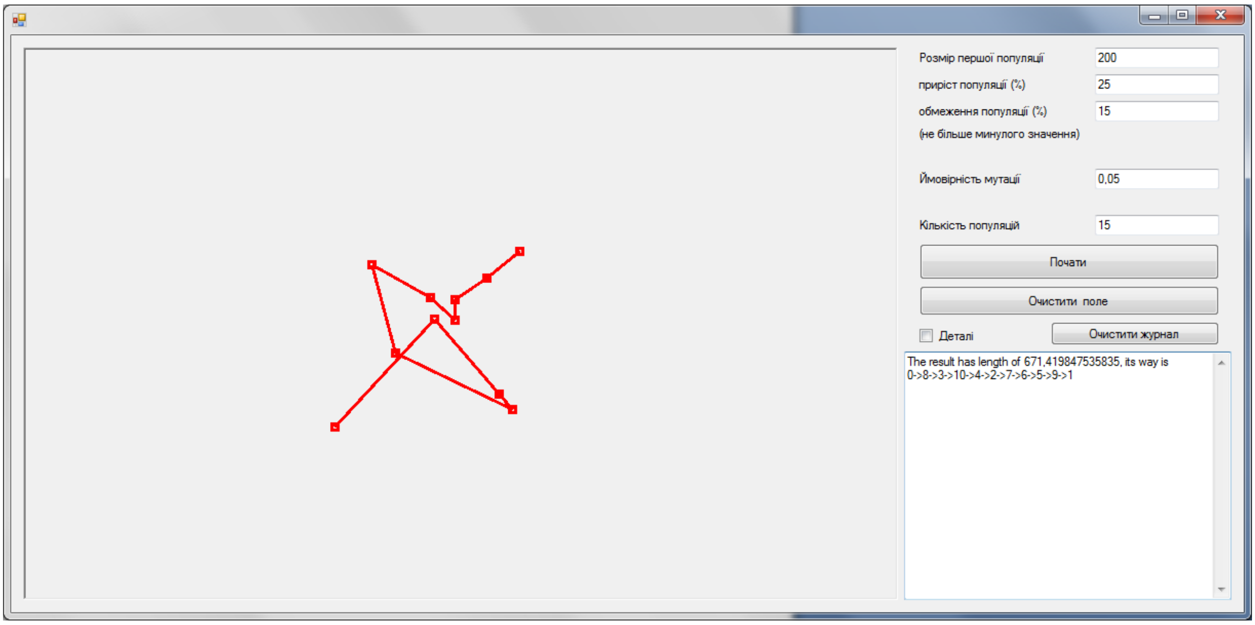
приріст популяції (%) 25

обмеження популяції (%) 15
(не більше минулого значення)

Ймовірність мутації 0,05

Кількість популяцій 10

```
The result has length of 722,526458716038, its way is
3->10->5->1->9->8->6->4->2->7->0
```



Розмір першої популяції 200

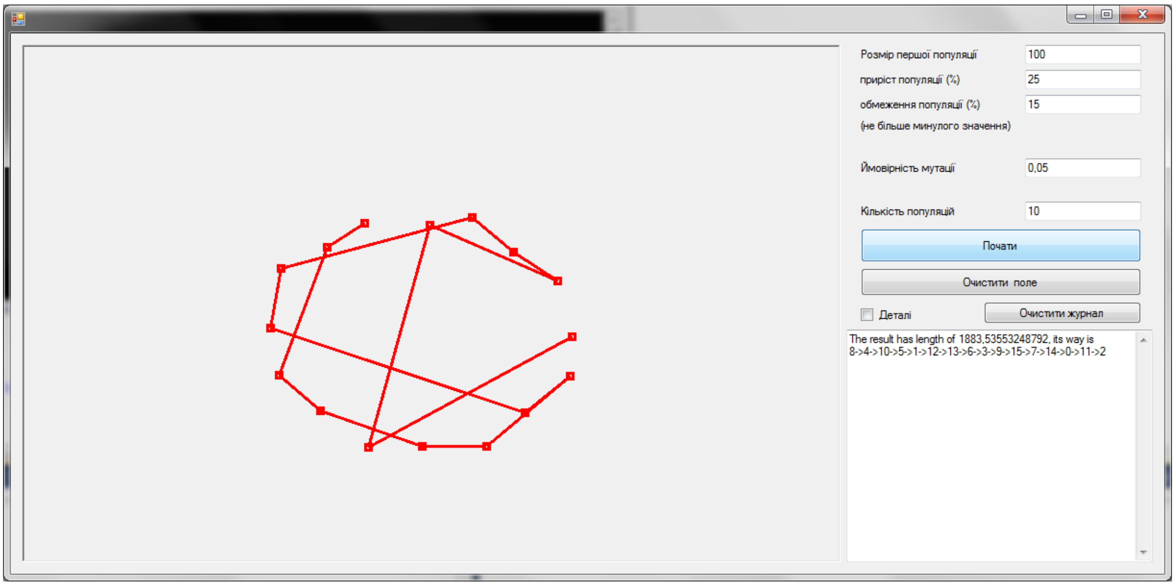
приріст популяції (%) 25

обмеження популяції (%) 15
(не більше минулого значення)

Ймовірність мутації 0,05

Кількість популяцій 15

```
The result has length of 671,419847535835, its way is
0->8->3->10->4->2->7->6->5->9->1
```



Розмір першої популяції 100

приріст популяції (%) 25

обмеження популяції (%) 15

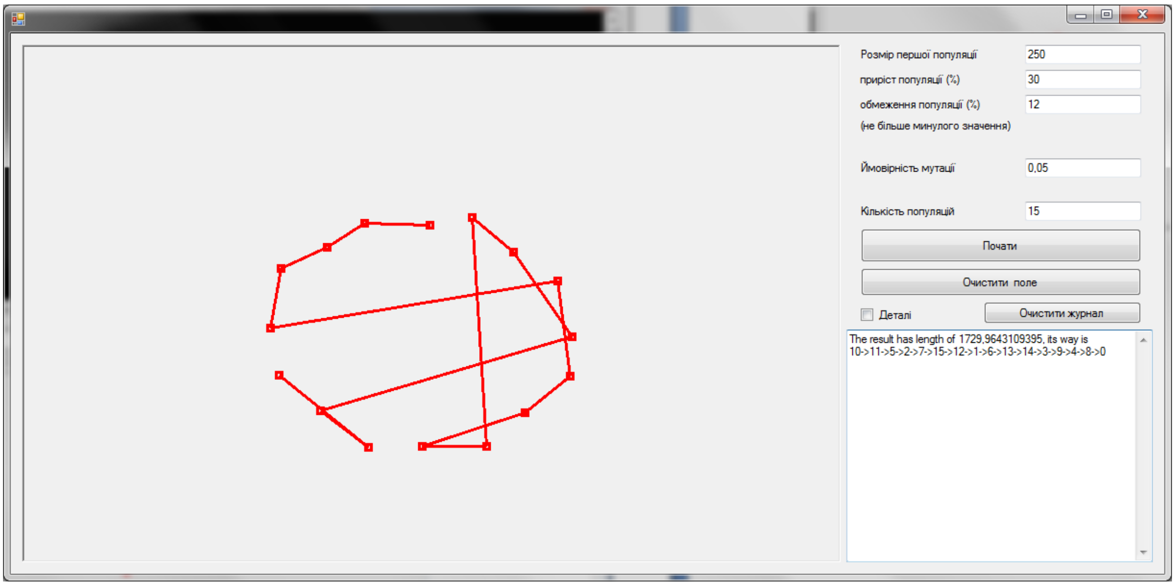
(не більше минулого значення)

Ймовірність мутації 0,05

Кількість популяцій 10

□ Деталі

```
The result has length of 1883,53553248792, its way is
8->4->10->5->1->12->13->6->3->9->15->7->14->0->11->2
```



Розмір першої популяції 250

приріст популяції (%) 30

обмеження популяції (%) 12

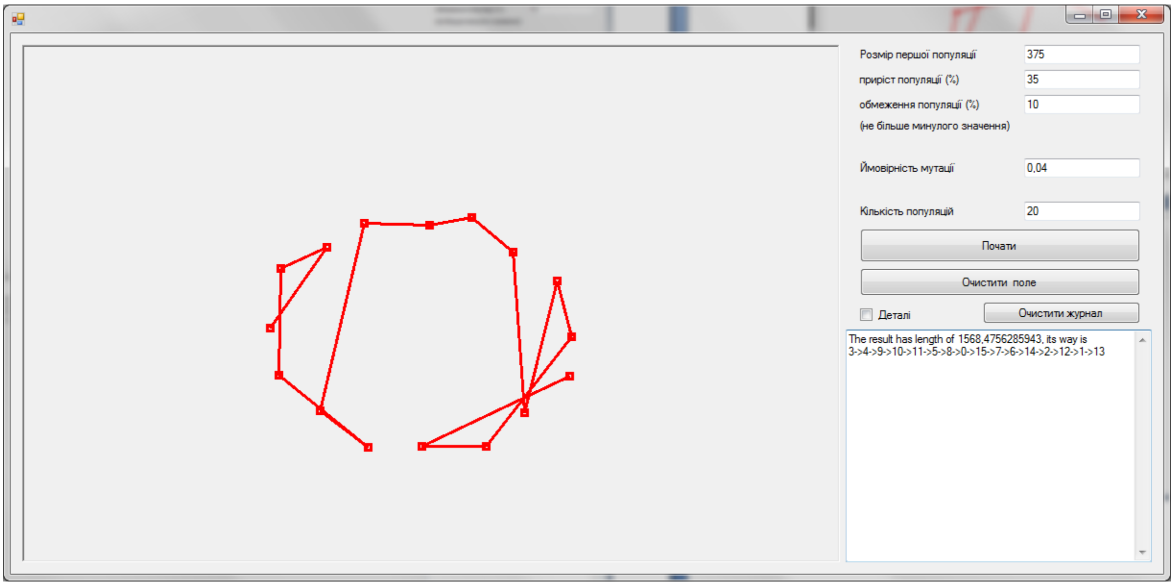
(не більше минулого значення)

Ймовірність мутації 0,05

Кількість популяцій 15

□ Деталі

```
The result has length of 1729,9643109395, its way is
10->11->5->2->7->15->12->1->6->13->14->3->9->4->8->0
```



Розмір першої популяції 375

приріст популяції (%) 35

обмеження популяції (%) 10

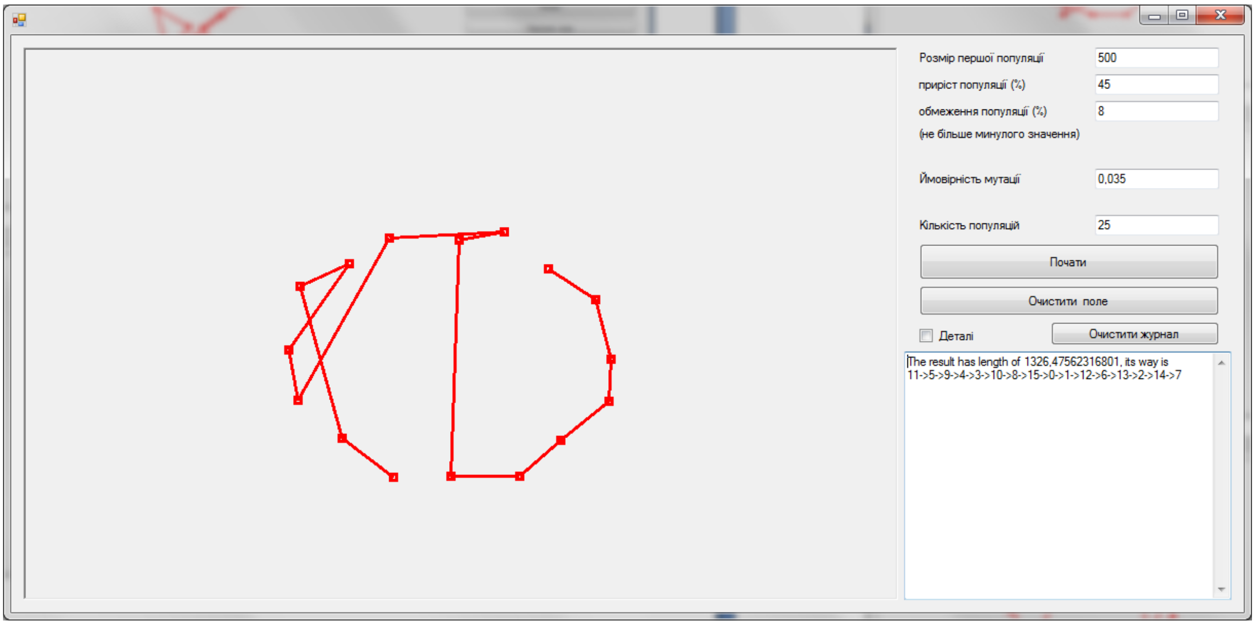
(не більше минулого значення)

Ймовірність мутації 0,04

Кількість популяцій 20

□ Деталі

```
The result has length of 1568,4756285943, its way is
3->4->9->10->11->5->8->0->15->7->6->14->2->12->1->13
```



Розмір першої популяції: 500

приріст популяції (%): 45

обмеження популяції (%): 8

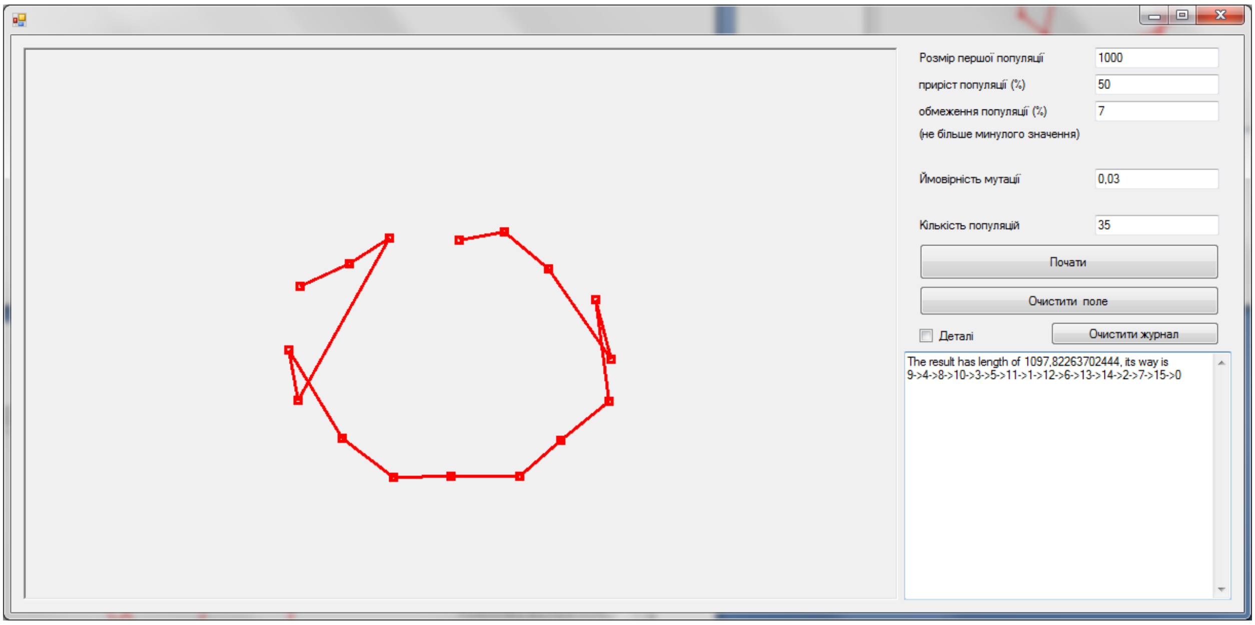
(не більше минулого значення)

Ймовірність мутації: 0,035

Кількість популяцій: 25

□ Деталі

```
The result has length of 1326,47562316801, its way is
11->5->9->4->3->10->8->15->0->1->12->6->13->2->14->7
```



Розмір першої популяції: 1000

приріст популяції (%): 50

обмеження популяції (%): 7

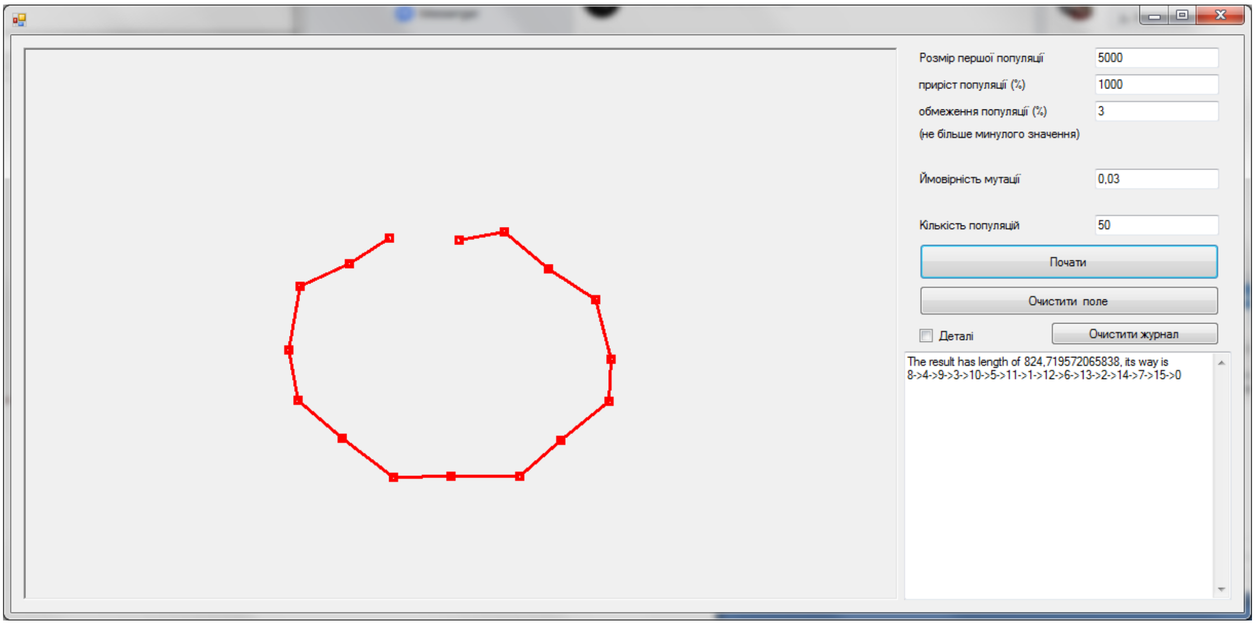
(не більше минулого значення)

Ймовірність мутації: 0,03

Кількість популяцій: 35

□ Деталі

```
The result has length of 1097,82263702444, its way is
9->4->8->10->3->5->11->1->12->6->13->14->2->7->15->0
```



Розмір першої популяції: 5000

приріст популяції (%): 1000

обмеження популяції (%): 3

(не більше минулого значення)

Ймовірність мутації: 0,03

Кількість популяцій: 50

□ Деталі

```
The result has length of 824,719572065838, its way is
8->4->9->3->10->5->11->1->12->6->13->2->14->7->15->0
```

Код програми:

```
             WayNeeded = false;
            panel1.Invalidate();
            Random a = new Random();
            int i,j,k;
            int NumberOfPop, NumberOfEnt, NewNumberOfEnt;
            double PosOfMut, LimOfPop, NewPop;
            // checking income
            if(Points.Count<2)
            {
                textBox2.Text += "Senseless work" + Environment.NewLine;
                return;
            }
            try
            {
                NumberOfPop = Convert.ToInt32(textBox1.Text);
                if (NumberOfPop <= 1 || NumberOfPop > 400)
                    throw new Exception();
            }
            catch
            {
                textBox2.Text += "Wrong number of generations! It shall be >1 and <=400"
+ Environment.NewLine;
                return;
            }
            try
            {
                NumberOfEnt = Convert.ToInt32(textBox4.Text);
                if (NumberOfEnt <= 0 || NumberOfEnt > 10000)
                    throw new Exception();
            }
            catch
            {
                textBox2.Text += "Wrong number of start entities. It shall be >0 and
<=10000" + Environment.NewLine;
                return;
            }
            try
            {
                PosOfMut = Convert.ToDouble(textBox3.Text);
                if (PosOfMut < 0 || PosOfMut >= 1)
                    throw new Exception();
            }
            catch
            {
                textBox2.Text += "Wrong value of mutation possibility. Its shall be in
limits from 0 to 1 and shall not be equal 1!" + Environment.NewLine;
                return;
            }
            try
            {
                    LimOfPop = 1+0.01*Convert.ToDouble(textBox6.Text);
                    if (LimOfPop < 1 || LimOfPop > 20)
                        throw new Exception();
            }
            catch
            {
                textBox2.Text += "Wrong number of population limit. It shall be in limits
from 0 to 1900%" + Environment.NewLine;
                return;
            }
            try
            {
                NewPop = 1 + 0.01 * Convert.ToDouble(textBox5.Text);
                if (NewPop < 1 || NewPop > 20 || NewPop<LimOfPop)
```

```
                    throw new Exception();
            }
            catch
            {
                textBox2.Text += "Wrong number of population growth. It shall be in
limits from 0 to 1900% and shouldnt be lesser then population limit" +
Environment.NewLine;
                return;
            }
            // creating  first generation
            int Temp;
            List<Entity> CurPopulation = new List<Entity>();
            List<int> ResToTakeFrom = new List<int>();//this thing is used to create a
new entity by taking random
            for (i = 0; i < Points.Count; i++)
            {
                ResToTakeFrom.Add(i);
            }
            Entity NewEntity = new Entity(new List<int>());
            List<int> ToTakeFrom = new List<int>();
            for (i = 0; i < NumberOfEnt; i++)
            {
                //ToTakeFrom = ResToTakeFrom;
                for (j = 0; j < Points.Count; j++)
                {
                    ToTakeFrom.Add(ResToTakeFrom[j]);
                }
                CurPopulation.Add(new Entity(new List<int>()));
                for (j = 0; j < Points.Count; j++)
                {
                    Temp = a.Next(ToTakeFrom.Count);
                    CurPopulation[i].Genes.Add(ToTakeFrom[Temp]);
                    ToTakeFrom.RemoveAt(Temp);
                }
                CurPopulation[i].Length=0;
                for (j = 1; j < Points.Count; j++)
                {
                    CurPopulation[i].Length +=
Math.Sqrt(Math.Pow(Points[CurPopulation[i].Genes[j]][0] -
Points[CurPopulation[i].Genes[j-1]][0], 2) +
Math.Pow(Points[CurPopulation[i].Genes[j]][1] - Points[CurPopulation[i].Genes[j-1]][1],
2));
                }

            }
            int NumberOfLastGen;
            int TA1, TA2;
            int TP1, TP2;
            double DT;
            for (int CurGeneration = 2; CurGeneration <= NumberOfPop; CurGeneration++)
            {
                NumberOfLastGen = CurPopulation.Count;
                //crossover
                for (NumberOfEnt = (int)Math.Floor(CurPopulation.Count * NewPop);
CurPopulation.Count < NumberOfEnt; )
                {
                    TA1 = a.Next(NumberOfLastGen);
                    TA2 = a.Next(NumberOfLastGen);
                    TP1 = 0;
                    TP2 = 0;
                    CurPopulation.Add(new Entity(new List<int>()));
                    for (j = 0; j < Points.Count; j++)
                    {
                        DT = a.NextDouble();
                        if (DT <= 0.5)
                        {
                            if (!CurPopulation[CurPopulation.Count -
1].Genes.Contains(CurPopulation[TA1].Genes[TP1]))
                            {
                                CurPopulation[CurPopulation.Count -
1].Genes.Add(CurPopulation[TA1].Genes[TP1]);
                                TP1++;
                            }
                            else
                            {
                                TP1++;
                                j--;
                                continue;
                            }
                        }
                        else
                        {
                            if (!CurPopulation[CurPopulation.Count -
1].Genes.Contains(CurPopulation[TA2].Genes[TP2]))
                            {
                                CurPopulation[CurPopulation.Count -
1].Genes.Add(CurPopulation[TA2].Genes[TP2]);
                                TP2++;
                            }
                            else
                            {
                                j--;
                                TP2++;
                                continue;
                            }
                        }
                       // CurPopulation[CurPopulation.Count-1].Genes.Add
                    }
                    CurPopulation[CurPopulation.Count - 1].Length = 0;
                    for (j = 1; j < Points.Count; j++)
                    {
                        CurPopulation[CurPopulation.Count - 1].Length +=
Math.Sqrt(Math.Pow(Points[CurPopulation[CurPopulation.Count - 1].Genes[j]][0] -
Points[CurPopulation[CurPopulation.Count - 1].Genes[j - 1]][0], 2) +
Math.Pow(Points[CurPopulation[CurPopulation.Count - 1].Genes[j]][1] -
Points[CurPopulation[CurPopulation.Count - 1].Genes[j - 1]][1], 2));
                    }
                }
                ////Mutations
                for (j = 0; j < CurPopulation.Count; j++)
                {
                    DT = a.NextDouble();
                    if (DT < PosOfMut)
                    {
                        TA1 = a.Next(Points.Count);
                        TA2 = a.Next(Points.Count);
                        k = CurPopulation[j].Genes[TA1];
                        CurPopulation[j].Genes[TA1] = CurPopulation[j].Genes[TA2];
                        CurPopulation[j].Genes[TA2] = k;
                        CurPopulation[j].Length = 0;
                        for (k = 1; k < Points.Count; k++)
                        {
                            CurPopulation[j].Length +=
Math.Sqrt(Math.Pow(Points[CurPopulation[j].Genes[k]][0] - Points[CurPopulation[j].Genes[k
- 1]][0], 2) + Math.Pow(Points[CurPopulation[j].Genes[k]][1] -
Points[CurPopulation[j].Genes[k-1]][1], 2));
                        }
                        j--;
                        continue;
                    }
                }
                    ////
                CurPopulation.Sort();
                for (NumberOfEnt = (int)Math.Floor(NumberOfLastGen * LimOfPop);
CurPopulation.Count > NumberOfEnt; )
                {
                    CurPopulation.RemoveAt(CurPopulation.Count-1);
                }
                if(checkBox1.Checked)
                {
                    textBox2.Text += "The best answer of step #" + CurGeneration + " has
length of " + CurPopulation[0].Length + ", its way is" + Environment.NewLine;
                    for (k = 0; k < Points.Count-1; k++)
                    {
                        textBox2.Text += CurPopulation[0].Genes[k] + "->";
                    }
                    textBox2.Text += CurPopulation[0].Genes[Points.Count - 1] +
Environment.NewLine;
                }
            }
            textBox2.Text += "The result has length of " + CurPopulation[0].Length + ",
its way is" + Environment.NewLine;
            for (k = 0; k < Points.Count - 1; k++)
            {
                textBox2.Text += CurPopulation[0].Genes[k] + "->";
            }
            textBox2.Text += CurPopulation[0].Genes[Points.Count - 1] +
Environment.NewLine;
            WayNeeded = true;
            PerfectWay = CurPopulation[0].Genes;
            panel1.Invalidate();
```