4 – Abramchuk Andrew

Description of action's program :

1) Set 30 random points on 10-d space;
2) Calculate distance matrix of 10-d space;
3) Set 30 random points on 2-d space;
4) Calculate distance matrix of 2-d space;
5) Matrix 10-d minus matrix 2-d;
6) Calculate fitness by summing values from matrix (5);
7) Repeat 3-6 100 times (1-st random generation);
8) Sorting (less – better);
9) … Tommorow

Code:

```
class Program
    {
        static void Main(string[] args)
        {
            Dimension obj = new Dimension();
            obj.GenerateMatrix();
            for (int i = 0; i < obj.numberOfPoints; i++)
            {
                for (int j = 0; j < obj.numberOfPoints; j++)
                    Console.Write("{0} ", obj._10DMatrixDistances[i, j]);
                Console.Write("\n");
            }
        }
}

class Dimension
    {
        public double[,] _10DMatrix;
        public double[,] _10DMatrixDistances;
        public double[,] _2DMatrix;
        public double[,] _2DMatrixDistances;
        public List<double[,]> List_2DMatrix;
        public double[,] fittnessMatrix;
        public List<double> GenerationFittnessList;
        public List<double> MinGenerationFittness;
        public List<double> AvgGenerationFittness;
        const int _10D = 10;
        const int _2D = 2;
        public readonly int numberOfPoints = 30;
        const int ChromosomesNumber = 100;

        #region  initialization and finding distance matrix

        public void GenerateMatrix()
        {
            Random rnd = new Random();
            _10DMatrix = new double[numberOfPoints, _10D];
            for(int i = 0; i < numberOfPoints; i++)
                for(int j = 0; j < _10D; j++)
                    _10DMatrix[i, j] = rnd.Next(0, 2001);
            _10DMatrix =  NormalizeRandom(_10DMatrix, _10D);
            _10DMatrixDistances = new double[numberOfPoints, numberOfPoints];
            for (int i = 0; i < numberOfPoints; i++)
                   for (int j = 0; j < numberOfPoints; j++)
```

```
                    _10DMatrixDistances[i, j] = FindDistanceND(i, j, _10D,
_10DMatrix);
            _10DMatrixDistances = NormalizeDistance(_10DMatrixDistances);
            GenerationFittnessList = new List<double>();
            List_2DMatrix = new List<double[,]>();
            int iter = 0;
            while (iter < 100)
            {
                Generate2DMatrix();
                FindFitness();
                List_2DMatrix.Add(_2DMatrix);
                iter++;
            }
            MinGenerationFittness = new List<double>();
            AvgGenerationFittness = new List<double>();
            MinGenerationFittness.Add(GenerationFittnessList.Min());
            AvgGenerationFittness.Add(GenerationFittnessList.Average());
        }

        void Generate2DMatrix()
        {
            Random rnd = new Random();
            _2DMatrix = new double[numberOfPoints, _2D];
            for (int i = 0; i < numberOfPoints; i++)
                for (int j = 0; j < _2D; j++)
                    _2DMatrix[i, j] = rnd.Next(0, 2001);
            _2DMatrix = NormalizeRandom(_2DMatrix, _2D);
            _2DMatrixDistances = new double[numberOfPoints, numberOfPoints];
            for (int i = 0; i < numberOfPoints; i++)
                for (int j = 0; j < numberOfPoints; j++)
                    _2DMatrixDistances[i, j] = FindDistanceND(i, j, _2D,
_2DMatrix);
            _2DMatrixDistances = NormalizeDistance(_2DMatrixDistances);
        }

        int FindDistanceND(int i , int j, int size, double[,] _10DMatr)
        {
            if (i == j) return 0;
            int num = 0;
            for (int k = 0; k < size; k++)
                num += (int)Math.Pow(_10DMatr[i, k] - _10DMatr[j, k], 2);
            return (int)Math.Sqrt(num);
        }

        double[,] NormalizeDistance(double[,] mat)
        {
            double max = mat.Cast<double>().Max();
            for(int i = 0; i < numberOfPoints; i++)
                for(int j = 0; j < numberOfPoints; j++)
                    mat[i, j] /= max;
            return mat;
        }

        double[,] NormalizeRandom(double[,] mat, int size)
        {
            for(int i = 0; i < numberOfPoints; i++)
                for(int j = 0; j < size; j++)
                {
                    mat[i, j] /= 1000;
                    if (mat[i, j] == 1) mat[i, j] = 0;
                    else if (mat[i, j] < 1) mat[i, j] = -mat[i, j];
                    else mat[i, j] -= 1;
                }
            return mat;
        }

        public void FindFitness()
```

```
            {
                fittnessMatrix = new double[numberOfPoints, numberOfPoints];
                double fittness = 0;
                for (int i = 0; i < numberOfPoints; i++)
                    for (int j = 0; j < numberOfPoints; j++)
                        fittnessMatrix[i, j] = _10DMatrixDistances[i, j] -
    _2DMatrixDistances[i, j];
                for (int i = 0; i < numberOfPoints; i++)
                    for (int j = 0; j < numberOfPoints; j++)
                        fittness += fittnessMatrix[i, j];
                GenerationFittnessList.Add(fittness);
            }

            #endregion

            public void MainCicle()
            {
                SortPopulation();
            }

            #region Sorting

            public void SortPopulation()
            {
                int[] keys = new int[ChromosomesNumber];
                for (int i = 0; i < ChromosomesNumber; i++) keys[i] = i;
                Array.Sort(GenerationFittnessList.ToArray(), keys);
                List<double[,]> sorted2DMatrix = new List<double[,]>();
                for (int i = 0; i < ChromosomesNumber; i++)
                    sorted2DMatrix.Add(FindKMatrix(keys[i]));
                List_2DMatrix = null;
                List_2DMatrix = sorted2DMatrix;
            }

            public double[,] FindKMatrix(int k)
            {
                for (int i = 0; i < numberOfPoints; i++)
                    if (i == k) return List_2DMatrix[i];
                return null;
            }

            #endregion

            public void UniformCrossover()
            {

            }

            public bool Uslovie()
            {
                if ( MinGenerationFittness.Last() == 0) return false;
                return true;
            }
}
```