4 exercise – Igor Kondraashuk

Task

From 10D graphic to 2D graphic.

Source code

Dot.cs

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

namespace lab3_siit_kondrashuk
{
    class Dots
    {
        public List<double> listX;
        public List<double> listY;
        public Dots(List<double> listX, List<double> listY)
        {
            this.listX = listX;
            this.listY = listY;
        }
        private void calculateFitness(List<List<double>> distanceTenD)
        {
            double fitness = 0;
            List<List<double>> distanceTwoD = new List<List<double>>();
            for(int i=0;i< distanceTwoD.Count;i++)
            return fitness;
        }
    }
}
```

GA.cs

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

namespace lab3_siit_kondrashuk
{
    class GA
    {
        Random random;
        List<Population> historyOfPopulation;
        public GA()
        {
            random = new Random();
            List<Dots> dots = new List<Dots>();
            for (int i = 0; i < 20; i++)
            {
                List<double> chromosomes = new List<double>();
                for (int j = 0; j < 20; j++)
                {
                    chromosomes.Add(random.NextDouble() % 2 -1);
                }
                dots.Add(new Dots(chromosomes));
            }
```

```
            //Sorting by fitness
            for (int i = 0; i < dots.Count; i++)
            {
                for (int j = dots.Count - 1; j > i; j--)
                {
                    if (dots[j].fitness > dots[j - 1].fitness)
                    {
                        Dots tempDot = dots[j];
                        dots[j] = dots[j - 1];
                        dots[j - 1] = tempDot;
                    }
                }
            }
            Population startPopulation = new Population(dots);
            historyOfPopulation = new List<Population>();
            historyOfPopulation.Add(startPopulation);
            int k = 0;
            while (!isReady(historyOfPopulation))
            {
                Thread.Sleep(10);
                historyOfPopulation.Add(getNextPupulation(historyOfPopulation[k]));
                k++;
            }
        }
        private Population getNextPupulation(Population parentPopulation)
        {
            List<Dots> childrenPopulationDots = new List<Dots>();

            for (int i = 0; i < 10; i++)
            {
                List<Dots> childrenDots = getChildren(
                    parentPopulation.dots[random.Next() % 10 + 10],
                    parentPopulation.dots[random.Next() % 10 + 10]
                    );
                childrenPopulationDots.AddRange(childrenDots);
            }
            //Sorting by fitness
            for (int i = 0; i < childrenPopulationDots.Count; i++)
            {
                for (int j = childrenPopulationDots.Count - 1; j > i; j--)
                {
                    if (childrenPopulationDots[j].fitness > childrenPopulationDots[j -
1].fitness)
                    {
                        Dots tempDot = childrenPopulationDots[j];
                        childrenPopulationDots[j] = childrenPopulationDots[j - 1];
                        childrenPopulationDots[j - 1] = tempDot;
                    }
                }
            }
            Population childrenPopulation = new Population(childrenPopulationDots);
            return childrenPopulation;
        }

        Boolean isReady(List<Population> historyOfPopulation)
        {
            if (historyOfPopulation.Count < 100)
                return false;
            else
            {
                for (int i = historyOfPopulation.Count - 100; i <
historyOfPopulation.Count; i++)
                {
                    if (historyOfPopulation[historyOfPopulation.Count -
100].avarageFitness != historyOfPopulation[i].avarageFitness)
```

```
                    return false;
                }
                return true;
            }
        }
        private List<Dots> getChildren(Dots father, Dots mother)
        {
            List<Dots> childrenDots = new List<Dots>();
            int pointCross = random.Next()%20;
            List<double> firstChromosomes = new List<double>();
            List<double> secondChromosomes = new List<double>();
            for (int j = 0; j < 20; j++)
            {
                if(j<pointCross)
                {
                    firstChromosomes.Add(father.chromosomes[j]);
                    secondChromosomes.Add(mother.chromosomes[j]);
                }
                else
                {
                    firstChromosomes.Add(mother.chromosomes[j]);
                    secondChromosomes.Add(father.chromosomes[j]);
                }
            }

            //Mutation
            for (int j = 0; j < childrenDots.Count; j++)
            {
                int prob = random.Next(0, 20);
                if (prob == 7)
                {
                    int number = random.Next(20);
                    childrenDots[j].chromosomes[number] = random.NextDouble() % 2 - 1;
                }
            }

            childrenDots.Add(new Dots(firstChromosomes));
            childrenDots.Add(new Dots(secondChromosomes));
            return childrenDots;
        }
    }
}
```

Population.cs

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

namespace lab3_siit_kondrashuk
{
    class Population
    {
        public List<Dots> dots;
        public double theBestFitness;
        public double avarageFitness;
        public Population(List<Dots> dogs)
        {
            this.dots = dogs;
            theBestFitness = getTheBestFitness(dogs);
            avarageFitness = getAvarageFitness(dogs);
        }
        public double getTheBestFitness(List<Dots> dogs)
        {
```

```
            double bestFitness = 9999;
            for (int i = 0; i < dogs.Count; i++)
                if (bestFitness > dogs[i].fitness)
                    bestFitness = dogs[i].fitness;
            return bestFitness;
        }
        public double getAvarageFitness(List<Dots> dogs)
        {
            double avarageFitness = 0;
            for (int i = 0; i < dogs.Count; i++)
                avarageFitness += dogs[i].fitness;
            avarageFitness /= dogs.Count;
            return avarageFitness;
        }
    }
}
```