Lucky dog

In this task we need to make 20 dogs (chromosomes with 1000 genes) run for the sausage. The field is matrix 1000x1000. Dog's house is in 500x500. There is only one sausage(in 800x800). Each gene of the dog told us in what direction to go in what step: 1 - up, 2 - down, 3 - right, 4 - left;

Fitness is the Manhattan distance from dog to sausage.

1. Random generation 20 dogs.
2. Sort dogs according to fitness.
3. Trunked selection 1/2.
4. Crossover.
5. Repeat while some dog can reach sausage.

We also need to visualize results with one point crossover and uniform crossover.

Code

Chromosom.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Chromosom {
    private List<Integer> genes = new ArrayList<Integer>();
    private static Random random = new Random();

    public Chromosom() {
        for(int i = 0; i < 1000; i++){
            genes.add(random.nextInt(4) + 1);
        }
    }

    public void mutate(){
        for(int i = 0; i < 1000; i++){
            if(random.nextDouble() < 0.01){
                genes.remove(i);
                genes.add(i, random.nextInt(4) + 1);
            }
        }
    }
```

```
    public static List<Chromosom> onePointCrossover(Chromosom first, Chromosom
second){
        int cut = random.nextInt(1000);

        List<Integer> firstChildGenes = new ArrayList<Integer>();
        List<Integer> secondChildGenes = new ArrayList<Integer>();

        for(int i = 0; i < 1000; i ++){
            if(i < cut){
                firstChildGenes.add(first.getGenes().get(i));
                secondChildGenes.add(second.getGenes().get(i));
            }
            else{
                firstChildGenes.add(second.getGenes().get(i));
                secondChildGenes.add(first.getGenes().get(i));
            }
        }


        Chromosom firstChild = new Chromosom();
        firstChild.setGenes(firstChildGenes);

        Chromosom secondChild = new Chromosom();
        secondChild.setGenes(secondChildGenes);

        List<Chromosom> children = new ArrayList<Chromosom>();
        children.add(firstChild);
        children.add(secondChild);

        return children;
    }

    public static List<Chromosom> uniformCrossover(Chromosom first, Chromosom
second){

        List<Integer> firstChildGenes = new ArrayList<Integer>();
        List<Integer> secondChildGenes = new ArrayList<Integer>();

        for(int i = 0; i < 1000; i ++){
            if(random.nextBoolean()){
                firstChildGenes.add(first.getGenes().get(i));
            }
            else{
                firstChildGenes.add(second.getGenes().get(i));
            }

            if(random.nextBoolean()){
                secondChildGenes.add(first.getGenes().get(i));
            }
            else{
                secondChildGenes.add(second.getGenes().get(i));
            }
        }

        Chromosom firstChild = new Chromosom();
        firstChild.setGenes(firstChildGenes);

        Chromosom secondChild = new Chromosom();
        secondChild.setGenes(secondChildGenes);

        List<Chromosom> children = new ArrayList<Chromosom>();
        children.add(firstChild);
        children.add(secondChild);

        return children;
    }
```

```
    public List<Integer> getGenes() {
        return genes;
    }

    public void setGenes(List<Integer> genes) {
        this.genes = genes;
    }

    public int fitness(){
        return Field.calcDogFitness(this);
    }
}
```

Population.java

```
import java.util.*;

public class Population {
    private List<Chromosom> dogs = new ArrayList<Chromosom>();
    private static Random random = new Random();

    public Population() {
        for (int i = 0; i < 20; i++) {
            dogs.add(new Chromosom());
        }
    }

    public List<Chromosom> getDogs() {
        return dogs;
    }

    public void setDogs(List<Chromosom> dogs) {
        this.dogs = dogs;
    }

    public void sort() {
        Collections.sort(dogs, new DogsComparator());
    }

    public Population getNextGenerationOnePointCrossover() {
        Population nextGeneration = new Population();
        nextGeneration.setDogs(this.getDogs());

        nextGeneration.sort();
        int size = nextGeneration.getDogs().size();
        for (int i = 0; i < size / 2; i++) {
            nextGeneration.getDogs().remove(nextGeneration.getDogs().size() - 1);
        }

        for (int i = 0; i < size / 4; i++) {
            List<Chromosom> children =
Chromosom.onePointCrossover(nextGeneration.getDogs().get(random.nextInt(getDogs().siz
e() - 1)), nextGeneration.getDogs().get(random.nextInt(getDogs().size() - 1)));
            for(Chromosom ch : children){
                ch.mutate();
            }
            nextGeneration.getDogs().addAll(children);
        }

        return nextGeneration;
    }

    public Population getNextGenerationUniformCrossover() {
        Population nextGeneration = new Population();
        nextGeneration.setDogs(this.getDogs());

        nextGeneration.sort();
```

```
        int size = nextGeneration.getDogs().size();
        for (int i = 0; i < size / 2; i++) {
            nextGeneration.getDogs().remove(nextGeneration.getDogs().size() - 1);
        }

        for (int i = 0; i < size / 4; i++) {
            List<Chromosom> children =
Chromosom.uniformCrossover(nextGeneration.getDogs().get(random.nextInt(getDogs().size
() - 1)), nextGeneration.getDogs().get(random.nextInt(getDogs().size() - 1)));
            for(Chromosom ch : children){
                ch.mutate();
            }
            nextGeneration.getDogs().addAll(children);
        }

        return nextGeneration;
    }

    class DogsComparator implements Comparator<Chromosom> {
        public int compare(Chromosom s1, Chromosom s2) {
            return Integer.compare(s1.fitness(), s2.fitness());
        }
    }

    public int getBestFitness(){
        return getDogs().get(0).fitness();
    }

    public double getAverageFitness(){
        double sum = 0.0;

        for(Chromosom chromosom : getDogs()){
            sum += chromosom.fitness();
        }

        return sum / getDogs().size();
    }
}
```

Field.java

```
import java.util.ArrayList;
import java.util.List;

public class Field {
    private static final int fieldHigh = 1000;
    private static final int fieldWeight = 1000;

    private static final int godsHouseXPos = 500;
    private static final int dogsHouseYPos = 500;

    private static final int sausageXPos = 800;
    private static final int sausageYPos = 800;

    public static int calcDogFitness(Chromosom dog){
        int currentXPos = godsHouseXPos;
        int currentYPos = dogsHouseYPos;

        for(int i : dog.getGenes()){
            if(i == 1){
                if(currentYPos + 1 < fieldHigh){
                    currentYPos++;
                }
                else{
                    currentYPos = 0;
                }
            }
```

```
if(i == 2){
    if(currentYPos - 1 < fieldHigh){
        currentYPos--;
    }
    else{
        currentYPos = fieldHigh;
    }
}

if(i == 3){
    if(currentXPos + 1 < fieldWeight){
        currentXPos++;
    }
    else{
        currentXPos = 0;
    }
}

if(i == 4){
    if(currentXPos - 1 < fieldWeight){
        currentXPos--;
    }
    else{
        currentXPos = fieldWeight;
    }
}
```

```
        int fitness = Math.abs((sausageXPos - currentXPos) + (sausageYPos -
currentYPos));

        return fitness;
    }

    public static List<Position> getDogTrace(Chromosom dog){
        List<Position> trace = new ArrayList<Position>();

        int currentXPos = godsHouseXPos;
        int currentYPos = dogsHouseYPos;

        for(int i : dog.getGenes()){
            if(i == 1){
                if(currentYPos + 1 < fieldHigh){
                    currentYPos++;
                }
                else{
                    currentYPos = 0;
                }
            }

            if(i == 2){
                if(currentYPos - 1 < fieldHigh){
                    currentYPos--;
                }
                else{
                    currentYPos = fieldHigh;
                }
            }

            if(i == 3){
                if(currentXPos + 1 < fieldWeight){
                    currentXPos++;
                }
                else{
                    currentXPos = 0;
                }
```

```
                }

                if(i == 4){
                    if(currentXPos - 1 < fieldWeight){
                        currentXPos--;
                    }
                    else{
                        currentXPos = fieldWeight;
                    }
                }

                trace.add(new Position(currentXPos, currentYPos));
            }

            return trace;
        }

    }
```

Position.java

```
public class Position {
    private int xPos;
    private int yPos;

    public Position(int xPos, int yPos) {
        this.xPos = xPos;
        this.yPos = yPos;
    }

    public int getxPos() {
        return xPos;
    }

    public void setxPos(int xPos) {
        this.xPos = xPos;
    }

    public int getyPos() {
        return yPos;
    }

    public void setyPos(int yPos) {
        this.yPos = yPos;
    }
}
```

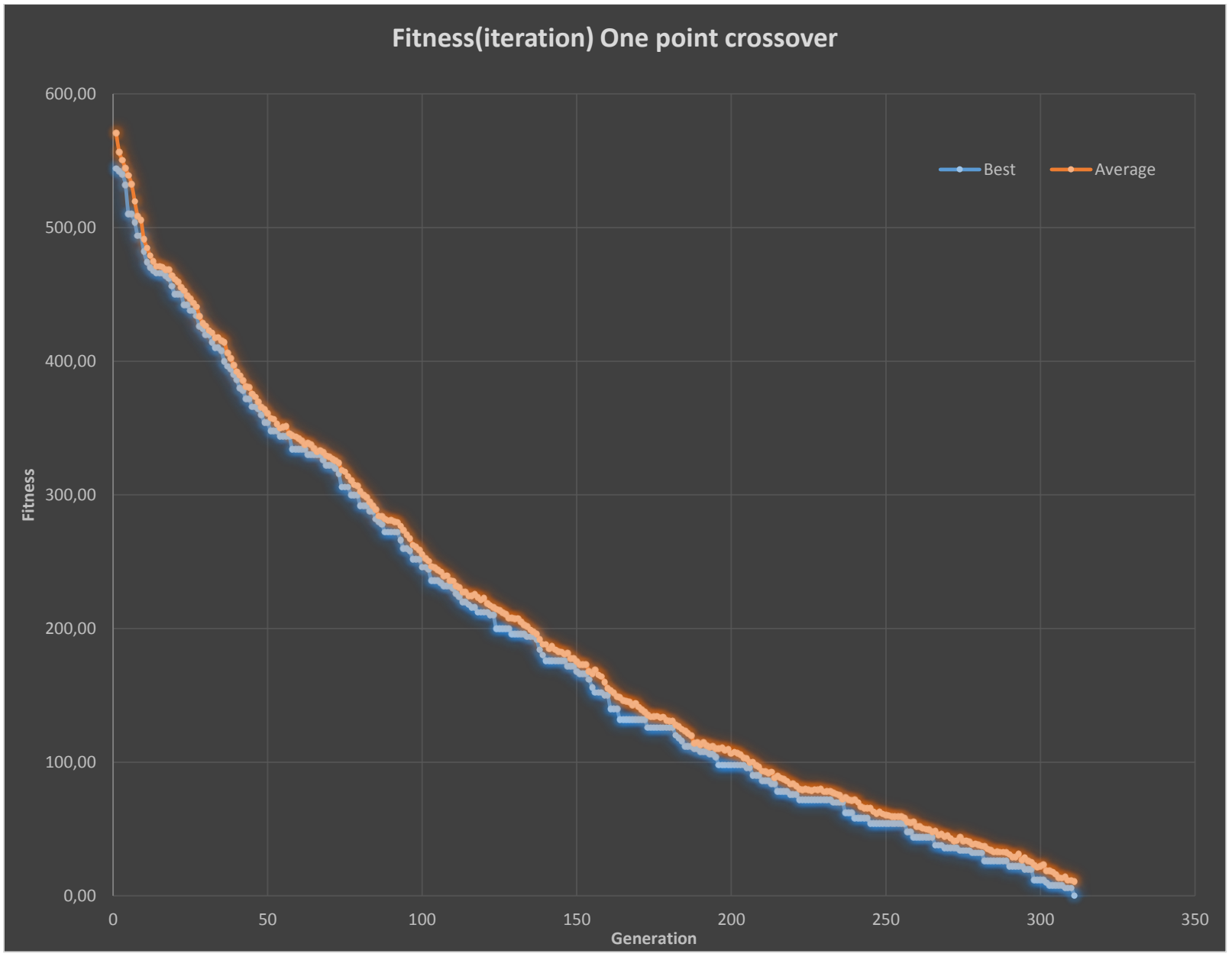
Main.java

```
import java.io.*;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.ArrayList;
import java.util.List;

import static java.nio.file.FileVisitResult.*;

public class Main {
    public static void main(String[] args) {
        List<Population> onePointCrossoverPopulations = new ArrayList<Population>();
        List<Population> uniformCrossoverPopulations = new ArrayList<Population>();

        Population onePointCrossoverCurrentPopulation = new Population();
        Population uniformCrossoverCurrentPopulation = new Population();
```

```
        PrintWriter onePointCr = null;
        PrintWriter uniformCr = null;

        PrintWriter onePointCr2 = null;
        PrintWriter uniformCr2 = null;
        try {
            new File("E:\\A_WORK\\5cource\\siit\\lab2\\result").mkdir();
            new
File("E:\\A_WORK\\5cource\\siit\\lab2\\result\\onePointCrossover").mkdir();
            new
File("E:\\A_WORK\\5cource\\siit\\lab2\\result\\uniformCrossover").mkdir();

            onePointCr = new
PrintWriter("result/onePointCrossover/onePointCrossover.txt", "UTF-8");
            onePointCr2 = new
PrintWriter("result/onePointCrossover/onePointCrossover2.txt", "UTF-8");

            uniformCr = new
PrintWriter("result/uniformCrossover/uniformCrossover.txt", "UTF-8");
            uniformCr2  = new
PrintWriter("result/uniformCrossover/uniformCrossover2.txt", "UTF-8");

            do {
                onePointCrossoverPopulations.add(onePointCrossoverCurrentPopulation);
                onePointCrossoverCurrentPopulation =
onePointCrossoverCurrentPopulation.getNextGenerationOnePointCrossover();

onePointCr.println(onePointCrossoverCurrentPopulation.getBestFitness());

onePointCr2.println(onePointCrossoverCurrentPopulation.getAverageFitness());
            } while (onePointCrossoverCurrentPopulation.getBestFitness() != 0);
            onePointCr.close();
            onePointCr2.close();

            do {
                uniformCrossoverPopulations.add(uniformCrossoverCurrentPopulation);
                uniformCrossoverCurrentPopulation =
uniformCrossoverCurrentPopulation.getNextGenerationUniformCrossover();

uniformCr.println(uniformCrossoverCurrentPopulation.getBestFitness());

uniformCr2.println(uniformCrossoverCurrentPopulation.getAverageFitness());
            } while (uniformCrossoverCurrentPopulation.getBestFitness() != 0);
            uniformCr.close();
            uniformCr2.close();

            for (int i = 0; i < onePointCrossoverPopulations.size(); i +=
onePointCrossoverPopulations.size() * 0.25) {
                if (i > onePointCrossoverPopulations.size() * 0.75) {
                    i = onePointCrossoverPopulations.size() - 1;
                }

                for (int d = 0; d < 3; d++) {
                    new
File("E:\\A_WORK\\5cource\\siit\\lab2\\result\\onePointCrossover\\onePointCrossoverTr
ace" + i).mkdir();
                    onePointCr = new PrintWriter("result/onePointCrossover/" +
"onePointCrossoverTrace" + i + "/dog" + d + ".txt", "UTF-8");

                    onePointCr.println(i);
                    for (Position p :
Field.getDogTrace(onePointCrossoverPopulations.get(i).getDogs().get(d))) {
                        onePointCr.println(p.getxPos() + "\t\t" + p.getyPos());
                    }

                    onePointCr.close();
```

```
                }
            }

            for (int i = 0; i < uniformCrossoverPopulations.size(); i +=
uniformCrossoverPopulations.size() * 0.25) {
                if (i > uniformCrossoverPopulations.size() * 0.75) {
                    i = uniformCrossoverPopulations.size() - 1;
                }

                for (int d = 0; d < 3; d++) {
                    new
File("E:\\A_WORK\\5cource\\siit\\lab2\\result\\uniformCrossover\\uniformCrossoverTrac
e" + i).mkdir();
                    uniformCr = new PrintWriter("result/uniformCrossover/" +
"uniformCrossoverTrace" + i + "/dog" + d + ".txt", "UTF-8");

                    uniformCr.println(i);
                    for (Position p :
Field.getDogTrace(uniformCrossoverPopulations.get(i).getDogs().get(d))) {
                        uniformCr.println(p.getxPos() + "\t\t" + p.getyPos());
                    }
                    uniformCr.close();
                }
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }


    }


}
```

One point crossover


Fitness(iteration) One point crossover
600,00
500,00
400,00
300,00
200,00
100,00
0,00
Fitness
0
50
100
150
200
250
300
350
Generation
Best
Average