DESIGN AND ANALYSIS OF ALGORITHMS

Project report

Submitted by :

Denis Aira-Benvenuti ES04122
Yanis Guerin Berrabeh ES04033
Thomas Seignour ES04098

UPM
UNIVERSITI PUTRA MALAYSIA
BERILMU BERBAKTI

Department of computer science
Semester 2 : 2022-2023

In this document, we will describe the program line per line and how it works. Function names have the same name as in pseudocodes presentation, so you can easily make a link.
To begin, we will explain the main function. It is not part of pseudocodes because it is not part of our algorithm, but as we need to implement it in java, we need to create the inputs and show the output. So, we will cover it in this document.

```java
public static void main(String[] args) {
    int N = 1000;
    List<Point2D> set = generateSet(N);

    Random rand = new Random();

    StraightLine line = new StraightLine( a: rand.nextInt( bound: 20)-10);

    System.out.println("Line equation\ny = " + line.getA() + " * x");

    Point2D closestPoint = closestPointToPlayer(set, N, line);

    if (closestPoint != null) {
        System.out.println("Closest point:\nx = " + closestPoint.getX() + "\ny = " + closestPoint.getY());
    } else {
        System.out.println("No obstacle on the line");
    }
}
```

This code first generate a set of 2D points of length N (set to 1000). The generateSet function is used, we will cover this below.
Then it generates a new line, with equation $y = a \times x$ randomly. It uses a class StraightLine, which only has an attribute a and a function getA() that gets line's a attribute. You can find the code in StraightLine.java
It prints the equation, then find the closestPoint using our algorithm.
Then, it tests the existence of such a point (we will see when it can happen). If there is one, it displays its X and Y coordinate. Otherwise, it displays that there is no obstacle on the line, which means there was no point found.

```java
public static List<Point2D> generateSet(int N) {
    Random rand = new Random();
    List<Point2D> set = new ArrayList<>();
    for (int i=0;i<N;i++) {
        Point2D point = new Point2D.Double( x: rand.nextDouble( bound: 100)-50, y: rand.nextDouble( bound: 100)-50);
        point.setLocation(Math.floor(point.getX()), Math.floor(point.getY()));
        set.add(point);
        // System.out.println("\nPoint " + i + ":\nx = " + set.get(i).getX() + "\ny = " + set.get(i).getY());
    }

    return set;
}
```

The generateSet function generates a new List of Point2D of length N.
We first declare a list of Point2D named set.
In the for loop, for each iteration, it first chooses a double (real number) between -50 and 50 for each coordinate (x and y), then floors them (which means it only takes the number before the comma. Ex: 7.89 becomes 7.0). Eventually, it adds the generated point in the set variable.
At the end of the for loop, it returns the set, containing all of the generated points.

```
public static Point2D closestPointToPlayer(List<Point2D> set, int N, StraightLine line) {
    List<Point2D> pointsOnLine = findPointsOnLine(set, N, line);

    if (pointsOnLine.size() == 0) {
        return null;
    } else {
        List<Point2D> pointsOnLineSorted = quicksort(pointsOnLine, l: 0, r: pointsOnLine.size()-1);
        return pointsOnLineSorted.get(0);
    }
}
```

This is our algorithm written in pseudocode. First, it finds all the points on the line with the findPointsOnLine algorithm.
If there is no point on the line, it just returns null. Otherwise, it sorts the points on the line by absolute X coordinate using the Quicksort algorithm. Then, it returns the first element of the sorted list. The idea behind is that, because we are on a linear expression $y = a \times x$ the point that is the closest to x=0,y=0 (which is player's position) is the point that with the lowest absolute X coordinate, as a is not a variable in this case, only a constant.
So the algorithm either returns the closest point to 0,0 or, if there is no point on the line, returns null

```
public static List<Point2D> findPointsOnLine(List<Point2D> set, int N, StraightLine line) {
    List<Point2D> pointsOnLine = new ArrayList<>();

    for (Point2D point : set) {
        if (point.getY() == line.getA() * point.getX()) {
            pointsOnLine.add(point);
        }
    }

    return pointsOnLine;
}
```

This algorithm finds all of the points on the line from a set of points and the line equation.
It first creates an empty list of 2D points named pointsOnLine.
Then it goes through every point in the provided set, and checks for each if the Y coordinate of the point is equal to the a constant of the line equation times the X coordinate of the point. If it does, it means the point is on the line, so it adds it to the pointsOnLine list.
At the end of the for loop, it returns pointsOnLine, which contains all of the points on the line. If there is no point on line, the list is returned empty.

```java
public static List<Point2D> quicksort(List<Point2D> A, int l, int r) {
    if (l < r) {
        int s = lomutoPartioning(A, l, r);
        quicksort(A, l, r: s-1);
        quicksort(A, l: s+1, r);
    }
    return A;
}
```

For this project, we decided to implement the quicksort algorithm which comes from the course. It is a recursive algorithm. We decided to use the lomutoPartitioning to make quicksort works.

```java
public static int lomutoPartioning(List<Point2D> A, int l, int r) {
    Point2D p = A.get(l);
    int s = l;
    for (int i=l+1;i<r+1;i++) {
        if (Math.abs(A.get(i).getX()) < Math.abs(p.getX())) {
            s++;
            swap(A, s, i);
        }
    }
    swap(A, l, s);
    return s;
}
```

Lomuto partitioning has been used for this code. The main part here is the if statement, which differs from what we are used to see. Indeed, we have a list of 2D points, and not just numbers. Also, we want to sort the array by absolute X coordinate of the points.
So, in this if statement, we take for each point we go through its X coordinate, and retrieve its absolute, and compare it to the absolute of the pivot X coordinate. If the pivot absolute X coordinate is greater than the observed point, s (pivot's index) increases, and we swap the items in position s and in position i (point's index)
At the end of the for loop, we undo the last swap. Then we return s, index of the pivot.

```java
public static void swap(List<Point2D> A, int index1, int index2) {
    Point2D tmp = A.get(index1);
    A.set(index1, A.get(index2));
    A.set(index2, tmp);
}
```

Swap function just swaps two items. It stores the first element to be swapped. Then it sets the second element to be swapped to the first index. Eventually, it sets the stored first element to second index.