Aufgabe 4: Schrebergärten

Team-ID: 00126

Team-Name: GSO

Bearbeiter dieser Aufgabe: Fabian Müller

Programmiersprache: Java

Inhaltsverzeichnis:

Lösungsidee

Zuerst habe ich mir Gedanken gemacht, was für eine Art Problem wir haben. Da wir für jeden Garten, der zu positionieren ist, jedes mal mehrere Möglichkeiten den Garten anzulegen haben, habe ich das Problem als nicht deterministisch in Polynomialzeit lösbar identifiziert. Nun habe ich mir die Frage gestellt, wie kann man die Gärten anordnen. Dazu habe ich mir überlegt, dass man sogenannte “entries” hat, also Stellen, an denen der nächste Garten angesetzt werden soll. Um das oben erwähnte weiter auszuführen:
wenn man einen Garten ansetzt, bekommt man immer weiter entries, damit auch immer mehr Möglichkeiten, wie die Gärten angeordnet seien können. Ich habe mir je 8 entries pro Garten überlegt: in jeder Ecke des (rechteckigen) Gartens jeweils an beiden, an der Ecke anliegenden Seiten. Hierbei ist zu beachten, dass der entry ggf. um seine Breite oder Höhe verschoben wird, damit der zu positionierende Garten nicht in den Garten, an den angesetzt wird, hineingeht. Um weitere Überlappungen zu verhindern, müssen alle entries entfernt werden, bei denen der zu positionierende Garten mit einem bereits positionierten Garten überlappt. Sind alle entries entfernt, die mit diesen Kriterien übereinstimmen, so kann der Garten dann noch an den verbleibenden entries angesetzt werden. Hierdurch entsteht wiederum ein neues Problem: wenn es mehr als einen entry gibt, dann weiß man gar nicht, mit welchem man weiterarbeiten muss und bis alle Gärten positioniert sind, weiß man nicht, welche Kombination das beste Ergebnis liefert.. Daher habe ich mich entschlossen mit jeder Möglichkeit weiterzuarbeiten; hierdurch entsteht die Problemkategorie wie oben bereits erwähnt. Man kann sehr leicht schauen, ob ein Ergebnis besser ist als ein anderes, nicht aber errechnen, welches Ergebnis das Beste ist. Jede eine Möglichkeit wird in Form einer Instanz erarbeitet. Jede Instanz gibt, wenn man einen Garten eingibt, alle möglichen Instanzen zurück. Im Fall dieser Aufgabe ist die Menge der zu positionierenden Gärten nie so groß, dass das Lösen der Aufgabe zeitlich nicht mehr vertretbar wäre.

Umsetzung

Ich habe meine Lösungsidee in rekursiven objektorientierten Code umgesetzt. Es existiert eine “Garden”-Klasse, die Größe und Breite enthält. Nun gibt es eine “PositionedGarden” Klasse, welche die Genaue Position des Gartens enthält. Zusätzlich gibt es eine “GardenInstance” Klasse, die eine Liste der positionierten Gärten enthält und eine next Methode, welche einen weiteren Garten wie in der Lösungsidee positioniert. Von einer “Calculator” Klasse, die alle Gärten als Liste enthält wird nun immer ein weiterer Garten eingespeist, bis keine Gärten mehr vorhanden sind. Die Fläche wird mit Vector2D und Area (beides Klassen) dargestellt, Vector2D ist, wie der Name impliziert, ein 2-dimensionaler Vektor. Eine Area eine Fläche. Die Area klasse hat eine “overlaps” Methode um die entries, wie oben beschrieben, zu überprüfen.

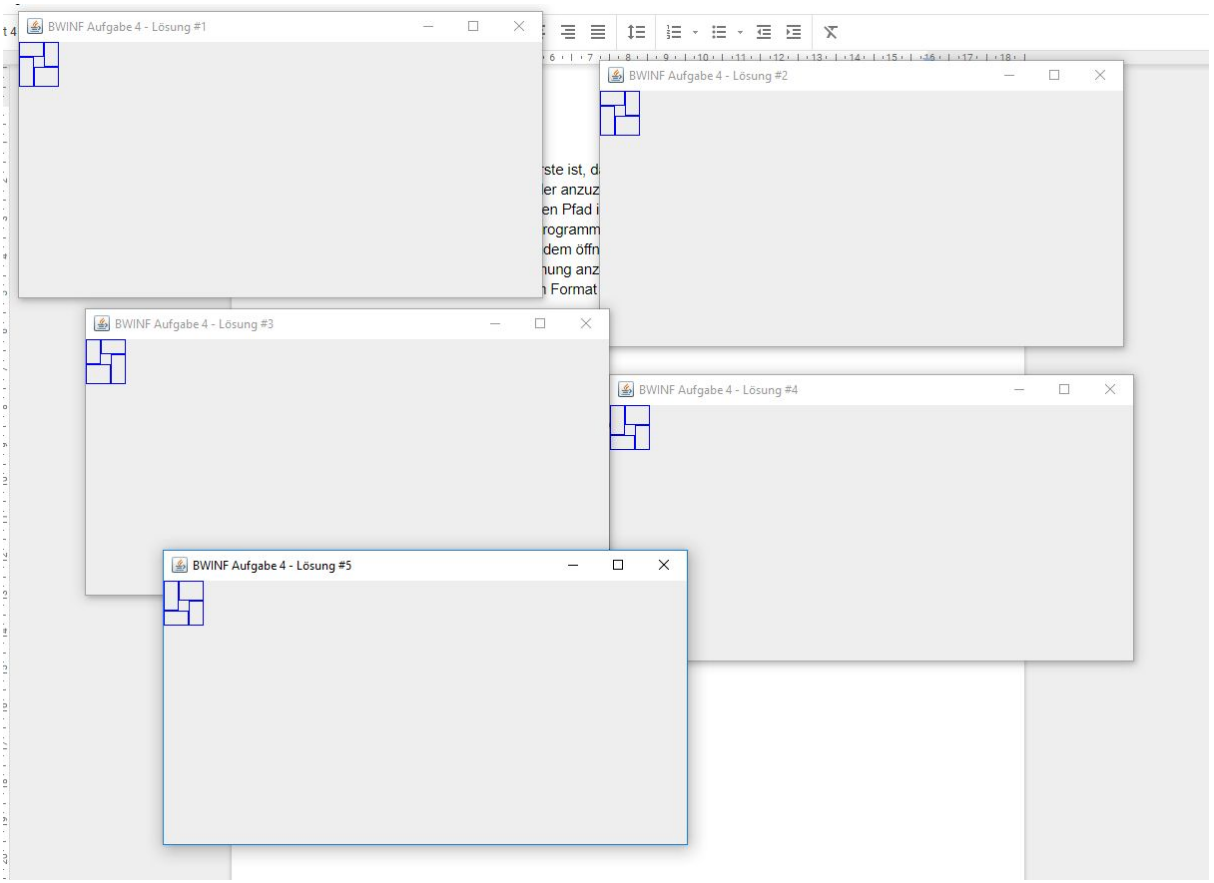
Bedienung

Es gibt zwei Möglichkeiten; die erste ist, das Programm mit den Argumenten “<PfadZurDatei>” und “<Anzahl der anzuzeigenden Lösungen>”, die zweite ist einfach das Programm zu starten und dann den Pfad in die Konsole einzugeben.
In beiden Fällen berechnet das Programm alle Möglichkeiten und gibt am Ende den niedrigsten Flächeninhalt aus. Zudem öffnen sich je nach Anzahl der anzuzeigenden Lösungen Fenster, die die Anordnung anzeigen. Die Datei, die eingegeben wird, muss in der ersten Zeile die Flächeninhalte im Format Breite x Länge, Breite2 x Länge2, … haben.

Beispiele

1. Anordnung: “bsp1.txt”

Verwendet man das erste Beispiel, so werden 5 verschiedene Anordnungen angezeigt, alle mit einem Flächeninhalt von 1800.



2. Die Grenzen: ‘bsp3.txt’

Das Programm filtert nicht nach Anzahl der Freiflächen, d. h. ggf. müssen mehrere Lösungen angeschaut werden, damit eine Lösung mit nur einer, zusammenhängenden, Freifläche angezeigt wid.

Quelltext

GardenInstance.java

```
public List<GardenInstance> next(Garden newGarden) {
    List<GardenInstance> gardenInstances = new ArrayList<>();

    List<Vector2D> allEntries = new ArrayList<>();

    for (Area area : this.areas) {
        int xa = area.a.getX();
        int xb = area.b.getX();
        int ya = area.a.getY();
        int yb = area.b.getY();

        allEntries.add(new Vector2D(xa - newGarden.getX(), ya));
        allEntries.add(new Vector2D(xb, ya));
        allEntries.add(new Vector2D(xa, yb));
        allEntries.add(new Vector2D(xb - newGarden.getX(), yb));
        allEntries.add(new Vector2D(xb, yb - newGarden.getY()));
        allEntries.add(new Vector2D(xa - newGarden.getY(), yb -
newGarden.getY()));
        allEntries.add(new Vector2D(xa, ya - newGarden.getY()));
        allEntries.add(new Vector2D(xb - newGarden.getX(), ya -
newGarden.getY()));
    }

    for (Vector2D entry : new ArrayList<>(allEntries)) {
        Area a = new Area(new Vector2D(entry.getX(), entry.getY()),
new Vector2D(entry.getX() + newGarden.getX(), entry.getY() +
newGarden.getY()));
        for (Area area : this.areas) {
            if (a.overlaps(area)) {
                allEntries.remove(entry);
                break;
            }
        }
```

```
    }

    if (allEntries.isEmpty() && this.gardens.isEmpty()) {
        allEntries.add(new Vector2D(0, 0));
    }

    for (Vector2D entry : allEntries) {
        List<PositionedGarden> nGardens = new
ArrayList<>(this.gardens);
        List<Area> nAreas = new ArrayList<>(this.areas);
        Area a = new Area(new Vector2D(entry.getX(), entry.getY()),
new Vector2D(entry.getX() + newGarden.getX(), entry.getY() +
newGarden.getY()));

        nGardens.add(new PositionedGarden(newGarden.getX(),
newGarden.getY(), a));
        nAreas.add(a);
        gardenInstances.add(new GardenInstance(nGardens, nAreas));
    }

    return gardenInstances;
}

public int getArea() {
    int x_min = 0;
    int y_min = 0;
    int x_max = 0;
    int y_max = 0;

    for (PositionedGarden garden : this.gardens) {
        if (garden.getPositionedArea().b.getX() > x_max) {
            x_max = garden.getPositionedArea().b.getX();
        }
        if (garden.getPositionedArea().b.getY() > y_max) {
            y_max = garden.getPositionedArea().b.getY();
        }
        if (garden.getPositionedArea().a.getX() < x_min) {
            x_min = garden.getPositionedArea().a.getX();
        }
        if (garden.getPositionedArea().a.getY() < y_min) {
            y_min = garden.getPositionedArea().a.getY();
        }
    }

    return (x_max - x_min) * (y_max - y_min);
}
```

```
public Area getAreaObject() {
    int x_min = 0;
    int y_min = 0;
    int x_max = 0;
    int y_max = 0;

    for (PositionedGarden garden : this.gardens) {
        if (garden.getPositionedArea().b.getX() > x_max) {
            x_max = garden.getPositionedArea().b.getX();
        }
        if (garden.getPositionedArea().b.getY() > y_max) {
            y_max = garden.getPositionedArea().b.getY();
        }
        if (garden.getPositionedArea().a.getX() < x_min) {
            x_min = garden.getPositionedArea().a.getX();
        }
        if (garden.getPositionedArea().a.getY() < y_min) {
            y_min = garden.getPositionedArea().a.getY();
        }
    }

    return new Area(new Vector2D(x_min, y_min), new Vector2D(x_max,
y_max));
}
```

Area.java

```
public boolean overlaps(Area area) {
    Vector2D other_a = area.a;
    Vector2D other_b = area.b;
    int x1 = a.getX();
    int x2 = b.getX();
    int y1 = a.getY();
    int y2 = b.getY();

    int ox1 = other_a.getX();
    int ox2 = other_b.getX();
    int oy1 = other_a.getY();
    int oy2 = other_b.getY();

    if (x1 < ox2 && x2 > ox1) {
        if(y1 < oy2 && y2 > oy1) {
            return true;
```

```
        }
    }

    return false;
}
```

Calculator.java

```
public List<GardenInstance> calculate() {
    List<GardenInstance> best = new ArrayList<>();

    List<GardenInstance> instances = new ArrayList<>();
    GardenInstance first = new GardenInstance(new ArrayList<>(), new
ArrayList<>());
    instances.add(first);
    Iterator<Garden> iterator = this.gardenList.iterator();

    while (iterator.hasNext()) {
        Garden garden = iterator.next();

        List<GardenInstance> nList = new ArrayList<>();

        for (GardenInstance instance : instances) {
            nList.addAll(instance.next(garden));
        }
        instances.clear();
        instances.addAll(nList);
        iterator.remove();
    }

    int area = Integer.MAX_VALUE;

    for (GardenInstance gi : instances) {
        if (gi.getArea() < area) {
            best.clear();
            area = gi.getArea();
            best.add(gi);
        } else if (gi.getArea() == area) {
            best.add(gi);
        }
    }

    return best;
}
```