Nandu

Turkins Kirils, Team-ID: 00271

27. Oktober 2023

Inhaltsverzeichnis

0 | Einleitung

Diese Dokumentation beschreibt die Entwicklung eines Programms, das die Lichtweitergabe in einem Nandu-System simuliert. Ein Nandu-System ist ein Netzwerk aus Lichtquellen und Lichtendpunkten, die durch Bausteine miteinander verbunden sind. Die Bausteine können boolesche Kombinationen ausführen je

nachdem welche Sensoren mit Licht bestrahlt sind. Das Ziel des Programms ist es, vorherzusagen welche Lichtendpunkte aus und welche an sind.

0.1 | Vorwort

In dieser Dokumentation kam GPT-4 minimal zum Einsatz, um die Grammatik zu berichtigen und auszubessern. Der Inhalt jedoch wurde komplett von einem Menschen verfasst.

Das Programm, von welchem sich diese Dokumentation handelt, wurde in Java 17 geschrieben mit der Verwendung von Gradle 8.2.1 als Build-Tool, um externe Frameworks wie JUnit für einfacheres und sauberes Testen des Programms zu verwenden. Außerdem erleichtert es dem BWINF-Team und den Lehrern, das Programm auszuführen, da alle modernen Entwicklungsumgebungen Build-Tools unterstützen und der Quellcode somit mit den meisten Entwicklungsumgebungen kompatibel ist.

Um die Umsetzung anschaulicher zu machen, werden in den Abschnitten *3.1 bis 3.3* kleine Quellcode-Ausschnitte präsentiert. Die vollständigen **relevanten** Quellcode-Komponenten sind in den Abschnitten *5.1 bis 5.3* zu finden.

1 | Problembeschreibung

Eine Herausforderung bei dieser Aufgabe ist, alle Elemente wie Bausteine und Lichtquellen/Lichtendpunkte als eine Einheit zu behandeln, um dieselbe Operation auf alle Elemente anzuwenden. Auch die roten Bausteine müssen mit anderen Bausteinen als eine Einheit gesehen werden, wobei sie sich durch den Unterschied auszeichnen, dass sie nur einen Sensor haben und sich somit von den anderen Massiv unterscheiden.

2 | Lösungsideen

2.1 | Gemeinsamkeit der Bausteine

Das Ziel ist, dass alle Bausteine zwei gemeinsame Eigenschaften haben. Sie können Licht empfangen und weiterleiten. Falls alle Bausteine diese Gemeinsamkeit haben, wird es sehr einfach Operationen auf alle Bausteine gleichzeitig anzuwenden.

2.2 | Lichtweitergabe

Um eine effiziente Lichtweitergabe zu erreichen, kann man alle Bausteine nacheinander besuchen und ihre Eigenschaft abfragen, ob sie eingeschaltet sind. Wenn sie eingeschaltet sind, geben sie eine Liste von Positionen zurück, an die das Licht weitergeleitet werden soll. Mit einer Liste können wir die Vererbung der Eigenschaft besser modellieren, für Lichtquellen, die nur eine neue Lichtposition weitergeben, und für Bausteine, die zwei Lichtpositionen weitergeben.

Vorerst werden die Positionen in einer weiteren Liste gespeichert und diese Liste wird jedem Baustein weitergegeben. Falls eine dieser Positionen in der Liste der Position einer der Sensoren entspricht, wird der Sensor vom Baustein aktiv.

2.3 | Modellierung

Die Idee ist, einen Baustein zu modellieren, der eine Position und einen Sensor hat. Dieser Baustein ist die Basis für eine Lichtquelle oder einen Lichtendpunkt, die beide nur eine Position benötigen. Aus diesem

Baustein kann man einen weiteren Doppel-Baustein ableiten, der zwei Positionen und zwei Sensoren hat. Dieser Doppel-Baustein erbt die Eigenschaften des ursprünglichen Bausteins.

Die drei verschiedenen Arten von Bausteinen wie der weisse, rote und blaue Baustein erben dan vom Doppel-Baustein, indem sie die Bedingung für die Lichtweitergabe überschreiben.

Bei dem roten Baustein, welcher eine Besonderheit hat, dass dieser nur einen Sensor hat, wird bei seiner Erzeugung eine Variable übergeben, welche dem Baustein berichtet, welcher der Sensoren bei der Bedingung berücksichtigt werden soll und welcher nicht.

3 | Umsetzung

3.1 | Vererbung

Wir verwenden eine abstrakte Klasse mit einer Position und einem Sensor als Basis für alle Bausteine. Für die weissen, roten und blauen Bausteine verwenden wir eine weitere abstrakte Klasse mit zwei Positionen und zwei Sensoren, die von der Basis erweitert.

Um das Licht zu erhalten, nehmen wir als Parameter eine Liste von Punkten und schauen ob einer dieser Punkte der Position des Bausteins entspricht, falls ja, aktivieren wir denn Sensor.

```
    public void activateSensor(List<Point> position) {
        for(Point pos : position){
            if (this.position.equals(pos)) {
                sensor = true;
                break;
            }
        }
    }
```

Damit diese Eigenschaft auch beim Doppel-Bausteinen funktioniert, überschreiben wir diese Eigenschaft, dass wir gleichzeitig prüfen, falls es nicht der ersten Position entspricht, ob es dann der zweiten Position des Bausteins entspricht.

```
for (Point pos : position) {
    if (super.position.equals(pos)) {
        super.sensor = true;
    } else if (additionalPosition.equals(pos)) {
        additionalSensor = true;
    }
}
```

3.2 | Lichtweitergabe

Wir verwenden eine abstrakte Methode für die Lichtweitergabe, die von den einzelnen Bausteinen implementiert wird. Diese Methode prüft, welche Sensoren seiner Bausteine aktiv sind und gibt anhand dieser Information eine Liste an Positionen zurück, wohin das neue Licht weitergehen soll.

```
public abstract Point[] activeLight();
```

3.2.1 | Weisser Baustein

Wir schauen beim weissen Baustein ganz einfach nach den beiden Sensoren. Wenn beide aktiv sind, geben wir keine Positionen zurück. Wenn einer nicht aktiv ist, geben wir zwei neue Positionen zurück, an denen das Licht weitergeleitet werden soll.

```
if (super.sensor && super.additionalSensor) {
    return new Point[0];
} else {
    Point leftSide = super.position;
    Point rightSide = super.additionalPosition;
    return new Point[]{
            new Point(leftSide.x, leftSide.y + 1),
        new Point(rightSide.x, rightSide.y + 1)
    };
}
```

3.2.2 | Roter Baustein

Beim roten Baustein haben wir eine Besonderheit, nämlich hat dieser nur einen Sensor. Dazu verwenden wir einen Konstruktor welcher bei der Erstellung dieses Bausteins aufgerufen wird. Dieser sagt aus ob der linke oder der rechte Sensor existiert. Sollte `true` weitergegeben werden, so ist der rechte Sensor anwesend, sonst ist der linke Sensor anwesend

```
public RedBox(boolean noSensor) {
        this.noSensor = noSensor;
}
```

Diese boolesche Variable wird dann in der Bedienung zur Lichtweitergabe verwendet.

```
// Die Variable "sensor" ist der linke Sensor und die Variable "additionalSensor"
ist der rechte Sensor
if (super.sensor && !noSensor || noSensor && super.additionalSensor) {
        [...]
} else {
        [...]
}
```

3.2.3 | Blauer Baustein

Der blaue Baustein ist recht simpel aufgebaut. Ist der linke Sensor aktiv, dann ist die Lichtweitergabe auf der linken Seite. Ist der rechte Sensor aktiv, dann ist die Lichtweitergabe auf der rechten Seite. Deshalb erstellen wir einen Array, der zwei Positionen beinhalten kann. Wenn ein Sensor deaktiviert ist, wird null als Position weitergegeben, das für keine Position steht.

```
Point[] activeLights = new Point[2];
activeLights[0] = super.sensor ? new Point(super.position.x, super.position.y + 1)
: null;
activeLights[1] = super.additionalSensor ? new Point(super.additionalPosition.x,
super.additionalPosition.y + 1) : null;
return activeLights;
```

3.3 | Erzeugung der Kombinationsmöglichkeiten

Die Logik hinter der Erzeugung der booleschen Kombinationen basiert auf der Umwandlung von Dezimalzahlen in boolesche Arrays. Jede Dezimalzahl kann durch eine Folge von Bits dargestellt werden, die 0 oder 1 sind. Um alle möglichen Kombinationen zu erzeugen, muss man alle Dezimalzahlen von 0 bis 2^n-1 durchgehen, wobei n die Länge der Kombination ist.

Zum Beispiel, wenn n = 3 ist, muss man die Zahlen von 0 bis 7 durchgehen. Jede Zahl wird dann in ein boolesches Array der Länge n umgewandelt, indem man die Bits von rechts nach links prüft. Dazu verwendet man Bitoperatoren, die bitweise Operationen auf den Zahlen ausführen.
Zum Beispiel, wenn die Zahl 5 ist, wird sie in Binärform als 101 geschrieben. Um sie in ein boolesches Array der Länge 3 umzuwandeln, muss man die Bits an den Positionen 0, 1 und 2 prüfen. Dazu verschiebt man die Zahl 1 um diese Positionen nach links und macht eine UND-Operation mit der Zahl 5. Das Ergebnis ist dann ein boolesches Array [true, false, true].

Dieses Verfahren wird für jede Zahl wiederholt und die resultierenden Arrays werden zu einer Liste hinzugefügt. Die Liste enthält dann alle booleschen Kombinationen.

4 | Beispiele

In diesem Abschnitt finden Sie alle relevanten Beispiele von der Ausgabe des Programms zur jeweiligen Eingabe.

4.1 | Gegebene Beispiele

Es wurden fünf Beispiele vom BWINF auf ihrer Website gegeben, der Link dazu hier: https://bwinf.de/bundeswettbewerb/42/1/

4.1.1 | Nandu_1

Link dieses Beispieles: https://bwinf.de/fileadmin/user_upload/nandu1.txt

Besonderheit: Alle Arten von Bausteinen sind in einem Beispiel angegeben

Eingabe:

```
X  Q1 Q2 X
X  W  W  X
r  R  R  r
X  B  B  X
X  W  W  X
X  L1 L2 X
```

Ausgabe:

```
| Q1 | Q2 | L1 | L2 |
| false | false | true | true |
| true | false | true | true |
| false | true | true | true |
| true | true | false | false |
```

4.1.2 | Nandu_2

Link dieses Beispieles: https://bwinf.de/fileadmin/user_upload/nandu2.txt

Besonderheit: Verzweigungen sind nach aussen angegeben, welche das Ergebnis beeinflussen

Eingabe:

```
X  X  Q1 Q2 X  X
X  X  B  B  X  X
X  X  W  W  X  X
X  r  R  R  r  X
r  R  W  W  R  r
X  W  W  B  B  X
X  X  B  B  X  X
X  X  L1 L2 X  X
```

Ausgabe:

```
| Q1 | Q2 | L1 | L2 |
| false | false | false | true |
| true | false | false | true |
| false | true | false | true |
| true | true | true | false |
```

4.1.3 | Nandu_3

Link dieses Beispieles: https://bwinf.de/fileadmin/user_upload/nandu3.txt

Besonderheit: Die Lichtquellen und die Lichtendpunkte sind nicht parallel angeordnet und es sind mehr Lichtendpunkte als Lichtquellen angegeben

Eingabe:

```
X  X  Q1 Q2 X  X  Q3 X
X  X  B  B  X  r  R  X
X  r  R  R  r  W  W  X
X  W  W  B  B  W  W  X
r  R  B  B  B  B  R  r
X  B  B  W  W  B  B  X
X  L1 L2 X  L3 L4 X  X
```

Ausgabe:

```
| Q1 | Q2 | Q3 | L1 | L2 | L3 | L4 |
| false | false | false | true | false | false | true |
| true | false | false | false | true | false | true |
| false | true | false | true | false | true | true |
| true | true | false | false | true | true | true |
| false | false | true | true | false | false | false |
| true | false | true | false | true | false | false |
| false | true | true | true | false | true | false |
| true | true | true | false | true | true | false |
```

4.1.4 | Nandu_4

Link dieses Beispieles: https://bwinf.de/fileadmin/user_upload/nandu4.txt

Besonderheit: Es sind mehr Lichtquellen als Lichtendpunkte angegeben

Eingabe:

```
X  X  Q1 Q2 Q3 Q4 X  X
X  r  R  B  B  R  r  X
X  X  W  W  W  W  X  X
X  r  R  B  B  R  r  X
r  R  W  W  B  B  R  r
X  W  W  W  W  W  W  X
X  X  B  B  X  X  X  X
X  X  L1 L2 X  X  X  X
```

Ausgabe:

```
| Q1 | Q2 | Q3 | Q4 | L1 | L2 |
| false | false | false | false | false | false |
| true | false | false | false | false | false |
| false | true | false | false | true | false |
| true | true | false | false | false | false |
| false | false | true | false | false | true |
| true | false | true | false | false | true |
| false | true | true | false | true | true |
| true | true | true | false | false | true |
| false | false | false | true | false | false |
| true | false | false | true | false | false |
| false | true | false | true | true | false |
| true | true | false | true | false | false |
| false | false | true | true | false | false |
| true | false | true | true | false | false |
| false | true | true | true | true | false |
| true | true | true | true | false | false |
```

4.1.5 | Nandu_5

Link dieses Beispieles: https://bwinf.de/fileadmin/user_upload/nandu5.txt

Besonderheit: Es sind viele Lichtquellen sowie Lichtendpunkte angegeben was zu insgesamt 36 verschiedenen Variationen führt, sprich 6^2

Eingabe:

```
X  X  X  X  X  Q1 Q2 X  X  Q3 Q4 X  X  Q5 Q6 X  X  X  X  X  X  X
X  X  X  X  X  R  r  X  X  r  R  X  X  R  r  X  X  X  X  X  X  X
X  X  X  X  r  R  R  r  r  R  R  r  r  R  R  r  X  X  X  X  X  X
X  X  X  X  W  W  B  B  W  W  B  B  W  W  B  B  X  X  X  X  X  X
X  X  X  r  R  B  B  B  B  B  B  B  B  B  B  R  r  X  X  X  X  X
X  X  r  R  B  B  X  B  B  B  B  B  B  B  B  X  R  r  X  X  X  X
X  r  R  B  B  R  r  B  B  B  B  B  B  B  B  r  R  R  r  X  X  X
X  X  B  B  W  W  B  B  X  B  B  B  B  B  B  W  W  B  B  X  X  X
X  r  R  W  W  W  W  R  r  W  W  W  W  W  W  W  W  W  W  X  X  X
X  W  W  B  B  W  W  X  B  B  W  W  B  B  W  W  B  B  R  r  X  X
r  R  B  B  B  B  R  r  X  B  B  B  B  B  B  B  B  B  B  R  r  X
X  B  B  B  B  B  B  R  r  B  B  B  B  B  B  B  B  B  B  W  W  X
r  R  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  R  r
L1 X  X  X  X L2  X  X  X  X L3  X  L4 X  X  X  X  X  X  X  X  L5
```

Ausgabe:

```
| Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | L1 | L2 | L3 | L4 | L5 |
| false | false | false | false | false | false | false | false | false | true |
false |
| true | false | false | false | false | false | true | false | false | true |
false |
| false | true | false | false | false | false | false | false | false | true |
false |
| true | true | false | false | false | false | true | false | false | true |
false |
| false | false | true | false | false | false | false | false | false | true |
false |
| true | false | true | false | false | false | true | false | false | true |
false |
| false | true | true | false | false | false | false | false | false | true |
false |
| true | true | true | false | false | false | true | false | false | true | false
|
| false | false | false | true | false | false | false | false | true | false |
false |
| true | false | false | true | false | false | true | false | true | false |
false |
| false | true | false | true | false | false | false | false | true | false |
false |
| true | true | false | true | false | false | true | false | true | false | false
|
| false | false | true | true | false | false | false | false | true | false |
false |
```

true	false	true	true	false	false	true	false	true	false	false
false	true	true	true	false	false	false	false	true	false	false
true	true	true	true	false	false	true	false	true	false	false
false	false	false	false	true	false	false	false	false	true	true
true	false	false	false	true	false	true	false	false	true	true
false	true	false	false	true	false	false	false	false	true	true
true	true	false	false	true	false	true	false	false	true	true
false	false	true	false	true	false	false	false	false	true	true
true	false	true	false	true	false	true	false	false	true	true
false	true	true	false	true	false	false	false	false	true	true
true	true	true	false	true	false	true	false	false	true	true	false	false	false	true	true	false	false	false	false	true	true
true	false	false	true	true	false	true	false	false	true	true
false	true	false	true	true	false	false	false	false	true	true
true	true	false	true	true	false	true	false	false	true	true	false	false	true	true	true	false	false	false	false	true	true
true	false	true	true	true	false	true	false	false	true	true	false	true	true	true	true	false	false	false	false	true	true
true	true	true	true	true	false	true	false	false	true	true	false	false	false	false	false	true	false	false	false	true	false
true	false	false	false	false	true	true	false	false	true	false
false	true	false	false	false	true	false	false	false	true	false
true	true	false	false	false	true	true	false	false	true	false
false	false	true	false	false	true	false	false	false	true	false
true	false	true	false	false	true	true	false	false	true	false
false	true	true	false	false	true	false	false	false	true	false
true	true	true	false	false	true	true	false	false	true	false
false	false	false	true	false	true	false	false	true	false	false
true	false	false	true	false	true	true	false	true	false	false
false	true	false	true	false	true	false	false	true	false	false
true	true	false	true	false	true	true	false	true	false	false

```
|
| false | false | true | true | false | true | false | false | true | false |
false |
| true | false | true | true | false | true | true | false | true | false | false
|
| false | true | true | true | false | true | false | false | true | false | false
|
| true | true | true | true | false | true | true | false | true | false | false |
| false | false | false | false | true | true | false | false | false | true |
true |
| true | false | false | false | true | true | true | false | false | true | true
|
| false | true | false | false | true | true | false | false | false | true | true
|
| true | true | false | false | true | true | true | false | false | true | true |
| false | false | true | false | true | true | false | false | false | true | true
|
| true | false | true | false | true | true | true | false | false | true | true |
| false | true | true | false | true | true | false | false | false | true | true
|
| true | true | true | false | true | true | true | false | false | true | true |
| false | false | false | true | true | true | false | false | false | true | true
|
| true | false | false | true | true | true | true | false | false | true | true |
| false | true | false | true | true | true | false | false | false | true | true
|
| true | true | false | true | true | true | true | false | false | true | true |
| false | false | true | true | true | true | false | false | false | true | true
|
| true | false | true | true | true | true | true | false | false | true | true |
| false | true | true | true | true | true | false | false | false | true | true |
| true | true | true | true | true | true | true | false | false | true | true |
```

4.2 | Eigene Beispiele

4.2.1 | Lichtendpunkt Verteilung

Besonderheit: Die Lichtendpunkte sind nicht am Ende des Feldes

Eingabe:

```
X  X  Q1 Q2 X  X  Q3 X
X  X  B  B  X  r  R  X
X  X  L1  R  r  W  W  X
X  W  W  B  B  W  W  X
r  R  B  B  B  B  R  r
X  B  B  W  W  L4  X  X
X  X  X  L2  L3 X  X  X
```

Ausgabe:

```
| Q1 | Q2 | Q3 | L1 | L2 | L3 | L4 |
| false | false | false | false | true | false | false |
| true | false | false | true | true | false | false |
| false | true | false | false | true | true | true |
| true | true | false | true | true | true | true |
| false | false | true | false | false | false | false |
| true | false | true | true | false | false | false |
| false | true | true | false | false | true | true |
| true | true | true | true | false | true | true |
```

4.2.2 | Unvollständige/Unkorrekte Angabe

Besonderheit:

- Lichtquellen und Lichtendpunkte sind nicht nummeriert
- Bausteine sind zusammengeschrieben
- Nicht alle Reihen haben dieselbe Anzahl an Feldern

Eingabe:

```
X  X  Q Q X  X  Q X
X  X  BB  X  r  R  X
X  X  L  R  r  W  W  X
X  W  W  B  B  W  W
r  R  B  B  B  B  R  r
X  BBWW  L  X  X
X  X  X  L  L
```

Wie dieses Beispiel vom Programm bearbeitet wird: Die Nummerierung ist für das Programm unwichtig, weil es in jedem Beispiel von links nach rechts erfolgt. Deshalb ignoriert das Programm diese Nummerierung, die ohnehin irrelevant ist und nummeriert die Elemente selbst von links nach rechts bei der Erstellung der Ausgabe Tabelle. Elemente wie Lichtquellen und Lichtendpunkte werden durchnummeriert von der links oberen Kante. Das Programm kommt mit zusammengeschriebenen Bausteinen zurecht, weil es in der Bearbeitung alle Leerzeichen entfernt und jeden Index als einzelnes Element behandelt. Die Anzahl der Felder pro Reihe muss nicht gleich sein, weil die Bausteine in einem einzigen Array mit ihren Positionen gespeichert werden. Bei der Lichtübergabe wird nach dem Baustein mit der entsprechenden Position gesucht. Wenn diese Position nicht existiert oder mit einem X markiert ist, wird das Licht ignoriert.

Ausgabe:

```
| false | false | false | false | true | false | false |
| true | false | false | true | true | false | false |
| false | true | false | false | true | true | true |
| true | true | false | true | true | true | true |
| false | false | true | false | false | false | false |
| true | false | true | true | false | false | false |
| false | true | true | false | false | true | true |
| true | true | true | true | false | true | true |
```

5 | Quellcode

Dieser Abschnitt enthält alle **wichtigen** Quellcode Komponenten, geschrieben in Java. Bei jedem Quellcode-Komponenten finden Sie den Pfad zur Klasse im Programm. Ausserdem enthalten viele dieser Komponenten keine Kommentare, da diese denn JavaDoc der Oberklasse erben oder besitzen ihre Beschreibung im Klassenkopf-JavaDoc. Eine weitere Anmerkung ist, dass Kommentare und Quellcode, einschließlich Variablen- und Klassennamen, auf Englisch verfasst sind. Dies entspricht der allgemeinen Konvention in der Programmierung. Der vollständige Quellcode ist im Programmordner angegeben.

5.1 | Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/Box.java*

```
public abstract class Box {

    protected boolean sensor;

    protected final Point position = new Point();

    /**
     * Sets the sensor to on, on given position of the box
     */
    public void activateSensor(List<Point> position) {
        for(Point pos : position){
            if (this.position.equals(pos)) {
                sensor = true;
                break;
            }
        }
    }

    public void activateSensor(boolean activate) {
        sensor = activate;
    }

    /**
     * Sets the position of the box
     */
    public void setStartingPosition(int x, int y) {
        position.x = x;
        position.y = y;
    }

    public void resetSensor(){
        sensor = false;
    }

    public int getRow() {
        return position.y;
    }

    /**
     * Returns an array of points which are active with light
```

```
     */
    public abstract Point[] activeLight();
}
```

5.1.1 | Lichtquelle/Lichtendpunkt Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/SingleBox.java*

```
    @Override
    public Point[] activeLight() {
        if (!super.sensor) {
            return new Point[0];
        } else {
            return new Point[]{
                    new Point(position.x, position.y + 1)
            };
        }
    }
```

5.2 | Doppel Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/DoubleBox.java*

```
/**
 * Represents a double box which contains two positions including, two sensors and
two lights
 */
public abstract class DoubleBox extends Box {

    protected boolean additionalSensor;
    protected final Point additionalPosition = new Point();

    @Override
    public void activateSensor(List<Point> position) {
        for (Point pos : position) {
            if (super.position.equals(pos)) {
                super.sensor = true;
            } else if (additionalPosition.equals(pos)) {
                additionalSensor = true;
            }
        }
    }

    @Override
    public void setStartingPosition(int x, int y) {
        super.position.x = x;
        super.position.y = y;
        additionalPosition.x = x + 1;
        additionalPosition.y = y;
    }

    @Override
```

```
    public void resetSensor() {
        super.resetSensor();
        additionalSensor = false;
    }
}
```

5.2.1 | Weisse Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/WhiteBox.java*

```
    @Override
    public Point[] activeLight() {
        if (super.sensor && super.additionalSensor) {
            return new Point[0];
        } else {
            Point leftSide = super.position;
            Point rightSide = super.additionalPosition;
            return new Point[]{
                    new Point(leftSide.x, leftSide.y + 1),
                    new Point(rightSide.x, rightSide.y + 1)
            };
        }
    }
```

5.2.2 | Rote Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/RedBox.java*

```
    private boolean noSensor;

    public RedBox(boolean noSensor) {
        this.noSensor = noSensor;
    }

    @Override
    public Point[] activeLight() {
        if (super.sensor && !noSensor || noSensor && super.additionalSensor) {
            return new Point[0];
        } else {
            Point leftSide = super.position;
            Point rightSide = super.additionalPosition;
            return new Point[]{
                new Point(leftSide.x, leftSide.y + 1),
                new Point(rightSide.x, rightSide.y + 1)
            };
        }
    }
```

5.2.3 | Blaue Bausteine

Pfad im Programmordner: *src/main/java/de/nandu/boxes/BlueBox.java*

```
    @Override
    public Point[] activeLight() {
        Point[] activeLights = new Point[2];
        activeLights[0] = super.sensor ? new Point(super.position.x,
super.position.y + 1) : null;
        activeLights[1] = super.additionalSensor ? new
Point(super.additionalPosition.x, super.additionalPosition.y + 1) : null;
        return activeLights;
    }
```

5.3 | Boolesche Kombinationen

Pfad im Programmordner: *src/main/java/de/nandu/controller/LightBoxField.java*

```
    // A method that returns a list of all possible boolean combinations of a
given length
    private static List<boolean[]> getBooleanCombinations(int n) {
        List<boolean[]> list = new ArrayList<>();
        // There are 2^n possible combinations
        int max = 1 << n;
        for (int i = 0; i < max; i++) {
            list.add(toBooleanArray(i, n));
        }
        return list;
    }

    // A helper method that converts a decimal number to a boolean array of a
given length
    private static boolean[] toBooleanArray(int n, int length) {
        boolean[] array = new boolean[length];
        for (int i = 0; i < length; i++) {
            array[i] = (n & (1 << i)) != 0;
        }
        return array;
    }
```