Tag 16 - The Floor Will Be Lava

Lösungshinweise Teil 1

- Interpretiere die Eingabedaten als char[][]-Array
- Lege zudem noch ein String[][]-Array an. In dieses kannst du pro Koordinate alle Richtungen speichern, mit denen du dich bereits über die Koordinate bewegt hast. Bsp. aktuell ist an der Koordinate "u" für upwards gespeichert. Jetzt betrittst du die Koordinate ein zweites Mal in Richtung downwards, also fügst du ein "d" an den String an → "ud". Später betrittst du die Koordinate erneut in Richtung downwards (du hast wahrscheinlich einen Loop und kannst in jedem Fall abbrechen!). Um den Abbruch zu erkennen, überprüfst du daher immer direkt zu Beginn jedes Schritts ob der Buchstabe der Richtung (hier "d" bereits in dem String der Koordinate vorhanden ist.
- Gehe Schritt für Schritt über das char[][]-Array entlang des Lichtstrahls. Nutze dafür eine rekursive Methode, z. B. `nextBeam()`. Rekursion ist nötig, da der Lichtstrahl an zwei möglichen Zeichen aufgesplittet werden kann. Das lässt sich mit einer einfachen Schleife nicht lösen. Wie funktioniert die Methode?
    - Die Methode benötigt nur drei Parameter (x- und y-Koordinate und die Richtung, in die der Lichtstrahl läuft (als char)).
    - Erzeuge in der Methode eine Endlosschleife mit `while(true) {…}`. Das tun wir, um immer alle Schritte in derselben Rekursiontiefe innerhalb der Schleife ausführen zu können **bis eine Aufsplittung des Lichtstrahls kommt**. Andernfalls kommt es zu einem Stack-Overflow, wenn man jeden Schritt direkt rekursiv ausführen würde.
    - Prüfe, ob die Koordinaten erlaubt sind (innerhalb des Spielfelds liegen) und ob man die Koordinate bereits in derselben Richtung betreten hat (siehe oberen Punkt). Bei Abbruch: `return` um die Endlosschleife abzubrechen.
    - Vermerke den Besuch mit der Richtung auf der Koordinate (siehe oberen Punkt).
    - Abschließend beginnen umfassende Verzweigungen. Im Lösungsvorschlag habe ich heute ausnahmsweise mal `switch-case` genutzt, anstatt `if-else`. Damit kann es etwas übersichtlicher werden. Man muss allerdings mit den `break`-Befehlen aufpassen!
    - Wenn der Lichtstrahl sich **nicht** aufsplittet, dann wird nur die Koordinate und falls nötig die Richtung angepasst, anschließend soll die Endlosschleife wieder von vorne beginnen.
    - Wenn der Lichtstrahl sich **aufsplittet**, dann müssen die zwei nächsten möglichen Wege rekursiv aufgerufen werden. Anschließend **muss** mit `return` die Endlosschleife abgeschlossen werden.

Lösungsvorschlag Teil 1

```
private String[][] lightPaths;
private char[][] tiles;
private int breite;
private int hoehe;

public int partOne() {
    breite = inputLines.get(0).length();
    hoehe = inputLines.size();
```

```
    // übertrage eingabedaten in char[][]
    tiles = new char[inputLines.get(0).length()][inputLines.size()];
    for (int y = 0; y < hoehe; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < breite; x++) {
            tiles[x][y] = line.charAt(x);
        }
    }

    // speichere hierin pro Zelle z. B. "ur" -> Licht ist in dieser Zelle
bereits nach oben (Up) und nach rechts (Right) gegangen
    lightPaths = new String[breite][hoehe];
    resetLightPaths();

    // starte die Rekursion, um dem Beam Step für Step zu folgen
    nextBeam(0, 0, 'r');

    // Lass die Anzahl an "energized tiles" berechnen.
    return getEnergizedTiles();
}

/**
 * x, y sind die Koordinaten des Tiles, auf das der Beam trifft
 * dir ist die Richtung AUS DER der Beam kommt/kam
 */
private void nextBeam(int xIn, int yIn, char dirIn) {
    int x = xIn;
    int y = yIn;
    char dir = dirIn;
    while (true) {

        // wenn Koordinate außerhalb von Spielfeld liegt -> abbrechen
        if (x < 0 || x >= breite || y < 0 || y >= hoehe) {
            return;
        }

        // wenn man auf der Koordinate bereits in dieselbe Richtung lief ->
abbrechen (loop)
        if (lightPaths[x][y].contains(String.valueOf(dir))) {
            return;
        }

        // den "Besuch" eintragen
        lightPaths[x][y] += dir;

        // entscheide je nach tile
        switch(tiles[x][y]) {
            case '.':
```

```
                switch(dir) {
                    case 'r':
                        x++;
                        break;
                    case 'l':
                        x--;
                        break;
                    case 'u':
                        y--;
                        break;
                    case 'd':
                        y++;
                        break;
                }
                break;
            case '-':
                switch(dir) {
                    case 'r':
                        x++;
                        break;
                    case 'l':
                        x--;
                        break;
                    case 'u':
                    case 'd':
                        int x1 = x-1;
                        int y1 = y;
                        nextBeam(x1, y1, 'l');
                        int x2 = x+1;
                        int y2 = y;
                        nextBeam(x2, y2, 'r');
                        return;
                }
                break;
            case '|':
                switch(dir) {
                    case 'r':
                    case 'l':
                        int x1 = x;
                        int y1 = y-1;
                        nextBeam(x1, y1, 'u');
                        int x2 = x;
                        int y2 = y+1;
                        nextBeam(x2, y2, 'd');
                        return;
                    case 'u':
                        y--;
                        break;
                    case 'd':
                        y++;
                        break;
```

```
                }
                break;
            case '/':
                switch(dir) {
                    case 'r':
                        y--;
                        dir = 'u';
                        break;
                    case 'l':
                        y++;
                        dir = 'd';
                        break;
                    case 'd':
                        x--;
                        dir = 'l';
                        break;
                    case 'u':
                        x++;
                        dir = 'r';
                        break;
                }
                break;
            case '\\':
                switch(dir) {
                    case 'r':
                        y++;
                        dir = 'd';
                        break;
                    case 'l':
                        y--;
                        dir = 'u';
                        break;
                    case 'd':
                        x++;
                        dir = 'r';
                        break;
                    case 'u':
                        x--;
                        dir = 'l';
                        break;
                }
                break;
        }
    }
}

private void resetLightPaths() {
    for (int y = 0; y < hoehe; y++) {
        for (int x = 0; x < breite; x++) {
```

```
            lightPaths[x][y] = "";
        }
    }
}

private int getEnergizedTiles() {
    int summe = 0;
    for (int y = 0; y < hoehe; y++) {
        for (int x = 0; x < breite; x++) {
            if (lightPaths[x][y].length() > 0) {
                summe++;
            }
        }
    }
    return summe;
}
```

Teil 2

Für Teil 2 müssen nur wenige Änderungen vorgenommen werden. In der Haupt-Methode müssen in 4 Schleifen der Reihe nach alle Möglichkeiten durchprobiert werden. Nach jedem Durchgang muss geprüft werden, ob damit das neue Maximum gefunden wurde. Wichtig: das String[][]-Array muss unbedingt jedes Mal zurückgesetzt werden.

Lösungsvorschlag Teil 2

```
public long partTwo() {
    int maxEnergy = 0;

    breite = inputLines.get(0).length();
    hoehe = inputLines.size();

    // übertrage eingabedaten in char[][]
    tiles = new char[inputLines.get(0).length()][inputLines.size()];
    for (int y = 0; y < hoehe; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < breite; x++) {
            tiles[x][y] = line.charAt(x);
        }
    }

    // speichere hierin pro Zelle z. B. "ur" -> Licht ist in dieser Zelle
bereits nach oben (Up) und nach rechts (Right) gegangen
    lightPaths = new String[breite][hoehe];

    // gehe in einer Schleife über alle Möglichkeiten in der obersten Zeile
    for (int sx = 0; sx < breite; sx++) {
        resetLightPaths();
        nextBeam(sx, 0, 'd');
```

```
        int e = getEnergizedTiles();
        if (e > maxEnergy) {
            maxEnergy = e;
        }
    }
    // ... in der linken Spalte
    for (int sy = 0; sy < hoehe; sy++) {
        resetLightPaths();
        nextBeam(0, sy, 'r');
        int e = getEnergizedTiles();
        if (e > maxEnergy) {
            maxEnergy = e;
        }
    }
    // ... in der unteren Zeile
    for (int sx = 0; sx < breite; sx++) {
        resetLightPaths();
        nextBeam(sx, hoehe-1, 'u');
        int e = getEnergizedTiles();
        if (e > maxEnergy) {
            maxEnergy = e;
        }
    }
    // ... in der rechten Spalte
    for (int sy = 0; sy < hoehe; sy++) {
        resetLightPaths();
        nextBeam(breite-1, sy, 'l');
        int e = getEnergizedTiles();
        if (e > maxEnergy) {
            maxEnergy = e;
        }
    }

    return maxEnergy;
}
```