Day 15: Warehouse Woes

Der heutige Tag lässt sich über Schleifen (Teil 1) und Rekursion (Teil 2) lösen. Weil es aber insgesamt einige Bedingungen auf einer zweidimensionalen Karte zu beachten gibt, zählt es zu den schwierigeren Rätseln.

Aus diesem Grund sind hier auch nur kommentierte Lösungsvorschläge zu sehen.

Lösungsvorschlag Teil 1

```
public void partOne() {
    int width = inputLines.get(0).length();
    int height = 0;
    // ermittle die Höhe der Karte
    for (String line: inputLines) {
        if (line.length()==0) {
            break;
        }
        height++;
    }

    robot = new int[2]; // als Instanzvariable - speichert x ([0]) und y
([1])
    map = new char[width][height];
    // Übertrage alle Chars vom Input in die Karte
    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < width; x++) {
            map[x][y] = line.charAt(x);
            if (line.charAt(x) == '@') { // markiere die Position des
Roboters
                robot[0] = x;
                robot[1] = y;
            }
        }
    }

    // Speichert die Bewegungsrichtung. [0] ist x- und [1] ist y-Richtung
    int[] movement = new int[2];
    for (int y = height+1; y < inputLines.size(); y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < line.length(); x++) {
            char c = line.charAt(x);
            if (c == '<') {
                movement = new int[]{-1, 0};
            } else if (c == '^') {
                movement = new int[]{0, -1};
            } else if (c == '>') {
                movement = new int[]{1, 0};
```

```
            } else {
                movement = new int[]{0, 1};
            }

            // rufe die separate Methode auf, die die Bewegung aller
beteiligten Blöcke durchführt, sofern möglich
            move(movement);
        }
    }

    int sumGPS = 0;
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            if (map[x][y] == 'O') {
                sumGPS += x + 100*y;
            }
        }
    }
    System.out.println(sumGPS);
}

private void move(int[] movement) {
    // finde die nächste freie Stelle in movement-richtung
    int[] free = robot;

    int steps = 1;
    char c = map[robot[0] + movement[0]*steps][robot[1] +
movement[1]*steps];
    while (c != '#') {
        if (c == '.') {
            free = new int[]{robot[0] + movement[0]*steps,robot[1] +
movement[1]*steps};
            break;
        }
        steps++;
        c = map[robot[0] + movement[0]*steps][robot[1] + movement[1]*steps];
    }

    // falls nichts zu bewegen (free wurde nicht aktualisiert)
    if (free[0] == robot[0] && free[1] == robot[1]) {
        return;
    }

    // sonst:
    // bewege alles von free zu robot in die movement-Richtung
    while (free[0] != robot[0] || free[1] != robot[1]) {
        map[free[0]][free[1]] = map[free[0]-movement[0]][free[1]-
movement[1]];
```

```
        // bewege free eins "zurück" in Richtung robot
        free[0] -= movement[0];
        free[1] -= movement[1];
    }

    // ehemaliges robot-feld leer-machen
    map[robot[0]][robot[1]] = '.';

    // robot koordinaten aktualisieren;
    robot[0] += movement[0];
    robot[1] += movement[1];
}
```

Lösungsvorschlag Teil 2

```
public void partTwo() {
    // Der Beginn ist sehr ähnlich wie Teil 1
    int width = inputLines.get(0).length()*2;
    int height = 0;
    for (String line: inputLines) {
        if (line.length()==0) {
            break;
        }
        height++;
    }

    robot = new int[2];
    map = new char[width][height];
    for (int y = 0; y < height; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < line.length(); x++) {
            map[x*2][y] = line.charAt(x);
            map[x*2+1][y] = line.charAt(x);
            if (line.charAt(x) == '@') { // @ = @.
                robot[0] = x*2;
                robot[1] = y;

                map[x*2+1][y] = '.';
            } else if (line.charAt(x) == 'O') { // O = []
                map[x*2][y] = '[';
                map[x*2+1][y] = ']';
            }
        }
    }

    int[] movement = new int[2];
    for (int y = height+1; y < inputLines.size(); y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < line.length(); x++) {
            char c = line.charAt(x);
```

```
            if (c == '<') {
                movement = new int[]{-1, 0};
            } else if (c == '^') {
                movement = new int[]{0, -1};
            } else if (c == '>') {
                movement = new int[]{1, 0};
            } else {
                movement = new int[]{0, 1};
            }

            // prüfe zunächst, ob sich der Roboter inklusive aller eventuell
beteiligten
            // Schachteln bewegen ließe.
            if (checkIfFree(robot[0], robot[1], movement)) {
                // Falls ja, dann bewege entsprechend alles, was dazugehört
                moveItAll(robot[0], robot[1], movement);

                // aktualisiere nach der Bewegung die Roboter-Koordinaten
                robot[0] += movement[0];
                robot[1] += movement[1];
            }
        }
    }

    int sumGPS = 0;
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            // zähle nur den Beginn einer Schachtel
            if (map[x][y] == '[') {
                sumGPS += x + 100*y;
            }
        }
    }
    System.out.println(sumGPS);
}

/**
 * "Bewegt" rekursiv die aktuelle Koordinate NACHDEM die davor-liegenden
 * Koordinaten bewegt wurden.
 */
private void moveItAll(int x, int y, int[] movement) {
    // Basisfall: eine freie Stelle -> es gibt nicht zu verschieben
    if (map[x][y] == '.') {
        return;
    }

    // die nächsten bewegen
    // links/rechts
    if (movement[1] == 0) {
```

```
        moveItAll(x+movement[0], y, movement);
    } else { // hoch/runter
        // wenn drüber/drunter der Beginn einer Schachtel ist
        if (map[x][y+movement[1]] == '[') {
            moveItAll(x, y+movement[1], movement);
            moveItAll(x+1, y+movement[1], movement);
        } // wenn drüber/drunter das Ende einer Schachtel ist
        else if (map[x][y+movement[1]] == ']') {
            moveItAll(x, y+movement[1], movement);
            moveItAll(x-1, y+movement[1], movement);
        } else {
            moveItAll(x, y+movement[1], movement);
        }
    }

    // das aktuelle bewegen
    map[x+movement[0]][y+movement[1]] = map[x][y];

    // die alte position des aktuellen löschen
    map[x][y] = '.';
}

/**
 * Prüft rekursiv, ob die aktuelle Koordinate bzw. die in Bewegungsrichtung
beteiligten
 * Koordinaten frei sind. Beachtet auch die neue Schachtel-Breite!
 */
private boolean checkIfFree(int x, int y, int[] movement) {
    if (map[x][y] == '.') {
        return true;
    } else if (map[x][y] == '#') {
        return false;
    }

    if (movement[1] == 0) { // left/right
        return checkIfFree(x+movement[0], y, movement);
    } else { // up/down
        // wenn drüber/drunter der Beginn einer Schachtel ist
        if (map[x][y+movement[1]] == '[') {
            return checkIfFree(x, y+movement[1], movement) &&
checkIfFree(x+1, y+movement[1], movement);
        } // wenn drüber/drunter das Ende einer Schachtel ist
        else if (map[x][y+movement[1]] == ']') {
            return checkIfFree(x-1, y+movement[1], movement) &&
checkIfFree(x, y+movement[1], movement);
        } else {
            return checkIfFree(x, y+movement[1], movement);
        }
    }
}
```