Tag 22 - Sand Slabs

Für den heutigen Tag gibt es nur Lösungsvorschläge

Lösungsvorschlag Teil 1

```
private int[] size = new int[]{0,0,0};
private int[][][] map; // the blocks are numbered (per input-line) -> one
number per map-coordinate
private HashMap<Integer,Brick> bricks;

public long partOne() {
    // finde max breite, tiefe, hoehe
    for (String line: inputLines) {
        int[] from =
Arrays.stream(line.split("~")[0].split(",")).mapToInt(Integer::parseInt).toA
rray();
        int[] to =
Arrays.stream(line.split("~")[1].split(",")).mapToInt(Integer::parseInt).toA
rray();
        for (int i = 0; i < 3; i++) {
            if (from[i] > size[i]) {
                size[i] = from[i];
            }
            if (to[i] > size[i]) {
                size[i] = to[i];
            }
        }
    }

    // erhöhe max jeweils um 1
    for (int i = 0; i < 3; i++) {
        size[i] = size[i] + 1;
    }

    map = new int[size[0]][size[1]][size[2]];
    // fülle Boden mit fiktiver Brick-Nummer -1
    for (int x = 0; x < size[0]; x++) {
        for (int y = 0; y < size[1]; y++) {
            map[x][y][0] = -1;
        }
    }

    bricks = new HashMap<Integer,Brick>();

    int brickNumber = 1;
    for (String line: inputLines) {
```

```
        Brick b = new Brick(brickNumber);

        int[] from =
Arrays.stream(line.split("~")[0].split(",")).mapToInt(Integer::parseInt).toA
rray();
        int[] to =
Arrays.stream(line.split("~")[1].split(",")).mapToInt(Integer::parseInt).toA
rray();

        // gehe über alle 3 Koordinaten:
        int x, y, z;
        for (int i = 0; i < 3; i++) {
            for (int k = from[i]; k <= to[i]; k++) {
                if (i == 0) {
                    map[k][from[1]][from[2]] = brickNumber;
                    b.addKoordinate(k, from[1], from[2]);
                } else if (i == 1) {
                    map[from[0]][k][from[2]] = brickNumber;
                    b.addKoordinate(from[0], k, from[2]);
                } else {
                    map[from[0]][from[1]][k] = brickNumber;
                    b.addKoordinate(from[0], from[1], k);
                }
            }
        }

        bricks.put(brickNumber, b);
        brickNumber++;
    }

    // alle Bricks herunterfallen lassen
    letFall();

    Set<Integer> deletionCandidates = new HashSet<Integer>();
    Set<Integer> dontDelete = new HashSet<Integer>();

    // jeder Bricks muss prüfen, welche Bricks unter ihm sind (wenn mehr als
1, dann diese aufnehmen als löschbar)
    for (Brick b: bricks.values()) {
        Set<Integer> bricksBelow = new HashSet<Integer>();
        for (int[] k: b.getKoordinaten()) {
            int below = map[k[0]][k[1]][k[2]-1];
            if (below > 0 && below != b.getBrickNumber()) {
                bricksBelow.add(below);
            }
        }

        // wenn bricksBelow > 1 sind, dann füge sie zu deletionCadidates
hinzu
```

```
        if (bricksBelow.size() > 1) {
            for (int bb: bricksBelow) {
                deletionCandidates.add(bb);
            }
        }
        // sonst zu dontDelete
        else {
            for (int bb: bricksBelow) {
                dontDelete.add(bb);
            }
        }
    }

    // wenn über einem Brick kein anderer ist, dann kann man ihn auch
löschen
    for (Brick b: bricks.values()) {
        boolean somethingAbove = false;
        for (int[] k: b.getKoordinaten()) {
            if (k[2] >= size[2]) {
                continue;
            }
            int above = map[k[0]][k[1]][k[2]+1];
            if (above != 0 && above != b.getBrickNumber()) {
                somethingAbove = false;
                break;
            }
        }
        if (!somethingAbove) {
            deletionCandidates.add(b.getBrickNumber());
        }
    }

    // lösche die nicht-zu-löschenden aus der Kandidaten-Liste
    for (int b: dontDelete) {
        deletionCandidates.remove(b);
    }

    return deletionCandidates.size();
}

private boolean checkIfBrickMayFall(int brickAbove) {
    Brick b = bricks.get(brickAbove);
    int brickNumber = b.getBrickNumber();
    for (int[] k: b.getKoordinaten()) {
        if (map[k[0]][k[1]][k[2]-1] != 0 && map[k[0]][k[1]][k[2]-1] !=
brickNumber) {
            return false;
        }
    }

    return true;
```

```
}

private void letBrickFall(int blockAbove) {
    Brick b = bricks.get(blockAbove);

    // alte Position von Karte löschen
    for (int[] k: b.getKoordinaten()) {
        map[k[0]][k[1]][k[2]] = 0;
    }

    // brick-Koordinaten aktualisieren
    b.moveAllDown();

    // in Karte aktualisieren
    for (int[] k: b.getKoordinaten()) {
        map[k[0]][k[1]][k[2]] = b.getBrickNumber();
    }
}

public int letFall() {
    Set<Integer> fallenBricks = new HashSet<Integer>();
    for (int z = 1; z < size[2]-1; z++) {
        for (int x = 0; x < size[0]; x++) {
            for (int y = 0; y < size[1]; y++) {
                // wenn es frei ist
                if (map[x][y][z] == 0) {
                    int blockAbove = map[x][y][z+1];
                    if (blockAbove != 0) {
                        if (checkIfBrickMayFall(blockAbove)) {
                            // dafür sorgen, dass diese Ebene erneut
durchsucht wird:
                            z = z-2;
                            x = size[0];
                            y = size[1];

                            letBrickFall(blockAbove);
                            fallenBricks.add(blockAbove);
                        }
                    }
                }
            }
        }
    }
    return fallenBricks.size();
}
```

Für Teil 2 muss man nur minimale Änderungen vornehmen:

Lösungsvorschlag Teil 2

```
private int[][][] copyArray(int[][][] m) {
    int[][][] newMap = new int[size[0]][size[1]][size[2]];
    for (int x = 0; x < size[0]; x++) {
        for (int y = 0; y < size[1]; y++) {
            for (int z = 0; z < size[2]; z++) {
                newMap[x][y][z] = m[x][y][z];
            }
        }
    }
    return newMap;
}

private HashMap<Integer,Brick> copyBricks(HashMap<Integer,Brick> bricks) {
    HashMap<Integer,Brick> newBricks = new HashMap<Integer,Brick>();
    for (int i = 1; i <= bricks.size(); i++) {
        Brick b = bricks.get(i);
        Brick nb = new Brick(i);
        for (int[] k: b.getKoordinaten()) {
            nb.addKoordinate(k[0], k[1], k[2]);
        }
        newBricks.put(i, nb);
    }
    return newBricks;
}

public long partTwo() {
    // finde max breite, tiefe, hoehe
    for (String line: inputLines) {
        int[] from =
Arrays.stream(line.split("~")[0].split(",")).mapToInt(Integer::parseInt).toA
rray();
        int[] to =
Arrays.stream(line.split("~")[1].split(",")).mapToInt(Integer::parseInt).toA
rray();
        for (int i = 0; i < 3; i++) {
            if (from[i] > size[i]) {
                size[i] = from[i];
            }
            if (to[i] > size[i]) {
                size[i] = to[i];
            }
        }
    }

    // erhöhe max jeweils um 1
    for (int i = 0; i < 3; i++) {
        size[i] = size[i] + 1;
    }

    map = new int[size[0]][size[1]][size[2]];
    // fülle Boden mit fiktiver Brick-Nummer -1
```

```
    for (int x = 0; x < size[0]; x++) {
        for (int y = 0; y < size[1]; y++) {
            map[x][y][0] = -1;
        }
    }

    bricks = new HashMap<Integer,Brick>();

    int brickNumber = 1;
    for (String line: inputLines) {
        Brick b = new Brick(brickNumber);

        int[] from =
Arrays.stream(line.split("~")[0].split(",")).mapToInt(Integer::parseInt).toA
rray();
        int[] to =
Arrays.stream(line.split("~")[1].split(",")).mapToInt(Integer::parseInt).toA
rray();

        // gehe über alle 3 Koordinaten:
        int x, y, z;
        for (int i = 0; i < 3; i++) {
            for (int k = from[i]; k <= to[i]; k++) {
                if (i == 0) {
                    map[k][from[1]][from[2]] = brickNumber;
                    b.addKoordinate(k, from[1], from[2]);
                } else if (i == 1) {
                    map[from[0]][k][from[2]] = brickNumber;
                    b.addKoordinate(from[0], k, from[2]);
                } else {
                    map[from[0]][from[1]][k] = brickNumber;
                    b.addKoordinate(from[0], from[1], k);
                }
            }
        }

        bricks.put(brickNumber, b);
        brickNumber++;
    }

    // alle Bricks herunterfallen lassen
    letFall();

    Set<Integer> deletionCandidates = new HashSet<Integer>();
    Set<Integer> dontDelete = new HashSet<Integer>();

    // jeder Bricks muss prüfen, welche Bricks unter ihm sind (wenn mehr als
1, dann diese aufnehmen als löschbar)
    for (Brick b: bricks.values()) {
```

```
        Set<Integer> bricksBelow = new HashSet<Integer>();
        for (int[] k: b.getKoordinaten()) {
            int below = map[k[0]][k[1]][k[2]-1];
            if (below > 0 && below != b.getBrickNumber()) {
                bricksBelow.add(below);
            }
        }

        // wenn bricksBelow > 1 sind, dann füge sie zu deletionCadidates
hinzu
        if (bricksBelow.size() > 1) {
            for (int bb: bricksBelow) {
                deletionCandidates.add(bb);
            }
        }
        // sonst zu dontDelete
        else {
            for (int bb: bricksBelow) {
                dontDelete.add(bb);
            }
        }
    }

    // wenn über einem Brick kein anderer ist, dann kann man ihn auch
löschen
    for (Brick b: bricks.values()) {
        boolean somethingAbove = false;
        for (int[] k: b.getKoordinaten()) {
            if (k[2] >= size[2]) {
                continue;
            }
            int above = map[k[0]][k[1]][k[2]+1];
            if (above != 0 && above != b.getBrickNumber()) {
                somethingAbove = false;
                break;
            }
        }
        if (!somethingAbove) {
            deletionCandidates.add(b.getBrickNumber());
        }
    }

    // lösche die nicht-zu-löschenden aus der Kandidaten-Liste
    for (int b: dontDelete) {
        deletionCandidates.remove(b);
    }

    // sichere den originalen Zustand
    int[][][] originalMap = copyArray(map);
    // sichere die originalen Bricks
    HashMap<Integer,Brick> originalBricks = copyBricks(bricks);
```

```
    int amountOfFallingBricks = 0;
    // für jeden Brick der NICHT ein delectionCandidate ist: lösche ihn aus der Map und lasse alles herunterfallen
    for (Brick b: bricks.values()) {
        if (deletionCandidates.contains(b.getBrickNumber())) {
            continue;
        }
        // stelle wieder die originale Vorlage bereit
        map = copyArray(originalMap);
        bricks = copyBricks(originalBricks);

        // lösche den Brick
        for (int[] k: b.getKoordinaten()) {
            map[k[0]][k[1]][k[2]] = 0;
        }

        amountOfFallingBricks += letFall();
    }

    return amountOfFallingBricks;
}
```