Sudoku-Löser

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Das Logikrätsel Sudoku dürften den meisten Menschen ein Begriff sein. Es geht darum, die Zahlen 1-9 in einem 9x9 Feld unterzubringen. Bei der Anordnung der Zahlen gilt: Weder in einer **Zeile**, noch in einer **Spalte**, noch in einem **3x3 Feld** darf zweimal dieselbe Zahl auftauchen.

Das erste der rechts zu sehenden Sudokus ist eine Vorlage, bei der bereits ein paar Zahlen vorgegeben sind. Das zweite Bild zeigt dann die dazugehörige Lösung. Alle roten Zahlen wurden auf Basis der vorgegebenen schwarzen Zahlen hinzugefügt.

Auch Sudokus lassen sich wunderbar mittels einfachem Backtracking durch Ausprobieren aller möglichen Kombinationen lösen. Tatsächlich gibt es bei Sudokus noch einige andere Möglichkeiten, eine Lösung auch deutlich "intelligenter" und zeitsparender zu finden. Grundsätzlich genügt aber das Backtracking.

Algorithmus

Vorüberlegungen:

- In der nachfolgend gegebenen

  Vorlage

  wird das Spielfeld in einem 81 Zeichen langen eindimensionalen int-Array namens `spielfeld` gespeichert. Das erleichtert wieder u. a. das rekursive Befüllen des nächsten Feldes. Man kann sich das Spielfeld bezüglich der Indizes also wie folgt vorstellen:

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62

63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

- Da es beim Sudoku in der Regel nur um das Finden einer einzigen Lösung geht, muss der rekursive Algorithmus aktiv abgebrochen werden. Ansonsten würde das Programm (wie bei den 8-Damen und den magischen Quadraten) immer noch weitere denkbare Lösungen suchen. Daher steht die boolean-Instanzvariable `loesungGefunden` zur Verfügung.
- Beim Start des Sudoku-Lösers werden leere Zellen (bei denen kein Startwert vorgegeben war) als `0` im `spielfeld` gespeichert.
- Achtet darauf, dass ihr Felder, die bereits vorbelegt sind, nicht überschreibt!

(A1)

1. Lade die Vorlage herunter, starte die JavaFX-Anwenung und mache dich rudimentär mit der Funktionsweise des Programms / der GUI vertraut. Du musst den bereits vorgegebenen Code für die GUI **nicht** nachvollziehen.
2. Du kannst in der GUI einzelne Zahlen vorgeben und dann auf den Knopf "Lösen" klicken. Daraufhin wird die Methode `loeseSudoku(0)` aufgerufen.
3. Mit dem Auswahlfeld, kannst du einige Sudokus laden.
4. Im Unterverzeichnis `sudokus` befinden sich zwei Sudoku-Dateien, die du selbst anpassen kannst, so dass du deine eigenen Sudokus in der GUI laden kannst.
5. Implementiere folgende Methoden:
    - `loeseSudoku(int feld)`, welche die grundlegende rekursive Methode darstellt.
    - `istFehlerfrei()`, welche das komplette Sudoku-Feld auf Fehler überprüft.

Nachfolgend findest du Tipps in Form von Pseudocode sowie Lösungsvorschläge.

Pseudocode Löse-Algorithmus

```
loeseSudoku(feld):
    wenn feld < 81:
        // wenn das Feld bereits vorbelegt ist: überspringen
        wenn spielfeld[feld] nicht 0:
            loeseSudoku(feld+1)
            return

        für i von 1 bis 9:
            setze die Zelle 'feld' in spielfeld auf i

            wenn das spielfeld fehlerfrei ist: // implementiere dazu die
separate methode istFehlerfrei()
                loeseSudoku(feld+1)
                wenn loesungGefunden ist true:
                    return

        setze die Zelle 'feld' in spielfeld auf 0 // keine der oberen Zahlen
war erfolgreich, es geht wieder eine Zelle zurück
```

```
    sonst:
        zeigeErgebnis()
        loesungGefunden = true
```

Tipp zur Methode `istFehlerfrei()`: Wenn du die Methode `istFehlerfrei()` testen möchtest, dann bietet sich an, ein im Code vorliegendes int[] Array zu nutzen, das du direkt überprüfen kannst. Du kannst das Array z. B. wie folgt anlegen:

```
private int[] spielfeld = new int[]{
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
    1,2,3,4,5,6,7,8,9,
};
```

Die Zahlen kannst du editieren wie du sie magst (das 9x9 Format muss eingehalten werden) und anschließend die Methode `istFehlerfrei()` aufrufen. Diese Methode muss dazu natürlich vorübergehend `public` werden...

Lösungsvorschlag loeseSudoku()

```
if (feld < 81) {
    // wenn das Feld bereits vorbelegt ist: überspringen
    if (spielfeld[feld] != 0) {
        loeseSudoku(feld+1);
        return;
    }

    for (int i = 1; i <= 9; i++) {
        // setze die nächste Zahl ins Feld
        spielfeld[feld] = i;

        // wenn das Spielfeld fehlerfrei ist dann gehe zum nächsten Feld
        if (istFehlerfrei()) {
            loeseSudoku(feld+1);
            if (loesungGefunden) {
                return;
            }
        }
    }
    // es ist keine der oberen Zahlen -> Feld auf 0 setzen
    spielfeld[feld] = 0;
}
else {
```

```
    zeigeErgebnis();
    loesungGefunden = true;
}
```

Lösungsvorschlag istFehlerfrei()

```
// überprüfe die Zeilen
for (int beginn = 0; beginn < 81; beginn += 9) {
    // überprüfe die jeweils nächsten 9 Felder auf Dopplungen
    for (int i = beginn; i < beginn+8; i++) {
        for (int j = i+1; j < beginn+9; j++) {
            // wenn zwei Felder identisch (aber nicht 0) sind, dann gibts
nen Fehler
            if (spielfeld[i] != 0 && spielfeld[i] == spielfeld[j]) {
                return false;
            }
        }
    }
}

// überprüfe die Spalten
for (int beginn = 0; beginn < 9; beginn++) {
    // überprüfe das jeweils 9-te Feld auf Dopplung
    for (int i = beginn; i < 72; i += 9) {
        for (int j = i+9; j < 81; j += 9) {
            if (spielfeld[i] != 0 && spielfeld[i] == spielfeld[j]) {
                return false;
            }
        }
    }
}

// überprüfe die 3x3-Blocks
int[] blockStartIndices = {
        0, 3, 6,
        27, 30, 33,
        54, 57, 60
    };
for (int beginn : blockStartIndices) {
    // speichere die werte eines blocks in einem temporären Array
    int[] block = new int[9];
    for (int i = 0; i < 9; i++) {
        int zeile = (i / 3) * 9;
        int spalte = i % 3;
        block[i] = spielfeld[beginn + zeile + spalte];
    }

    for (int i = 0; i < 8; i++) {
        for (int j = i + 1; j < 9; j++) {
```

```
            if (block[i] != 0 && block[i] == block[j]) {
                return false;
            }
        }
    }
}

return true;
```

From:
https://info-bw.de/ -

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:algorithmen:rekursion:backtracking:sudoku-loeser:start?rev=1740665183

Last update: **27.02.2025 14:06**