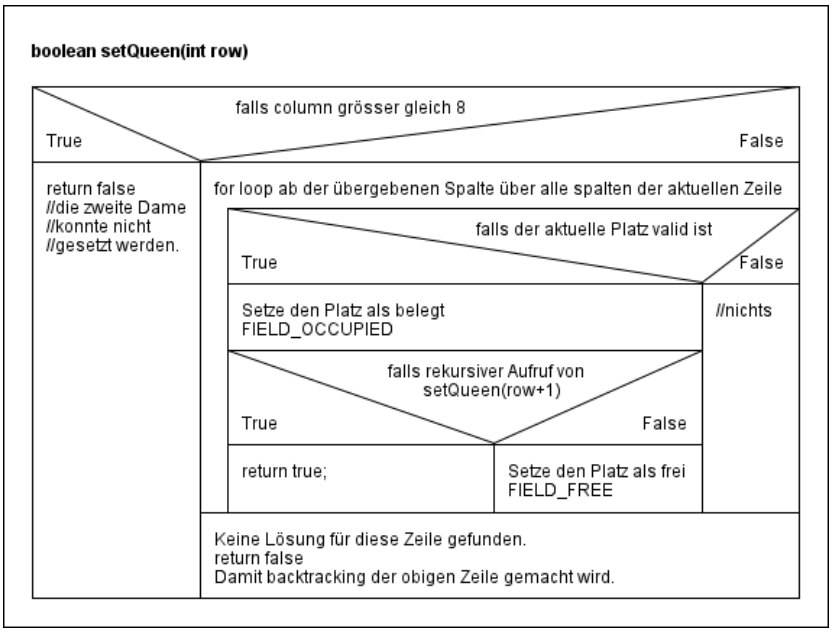
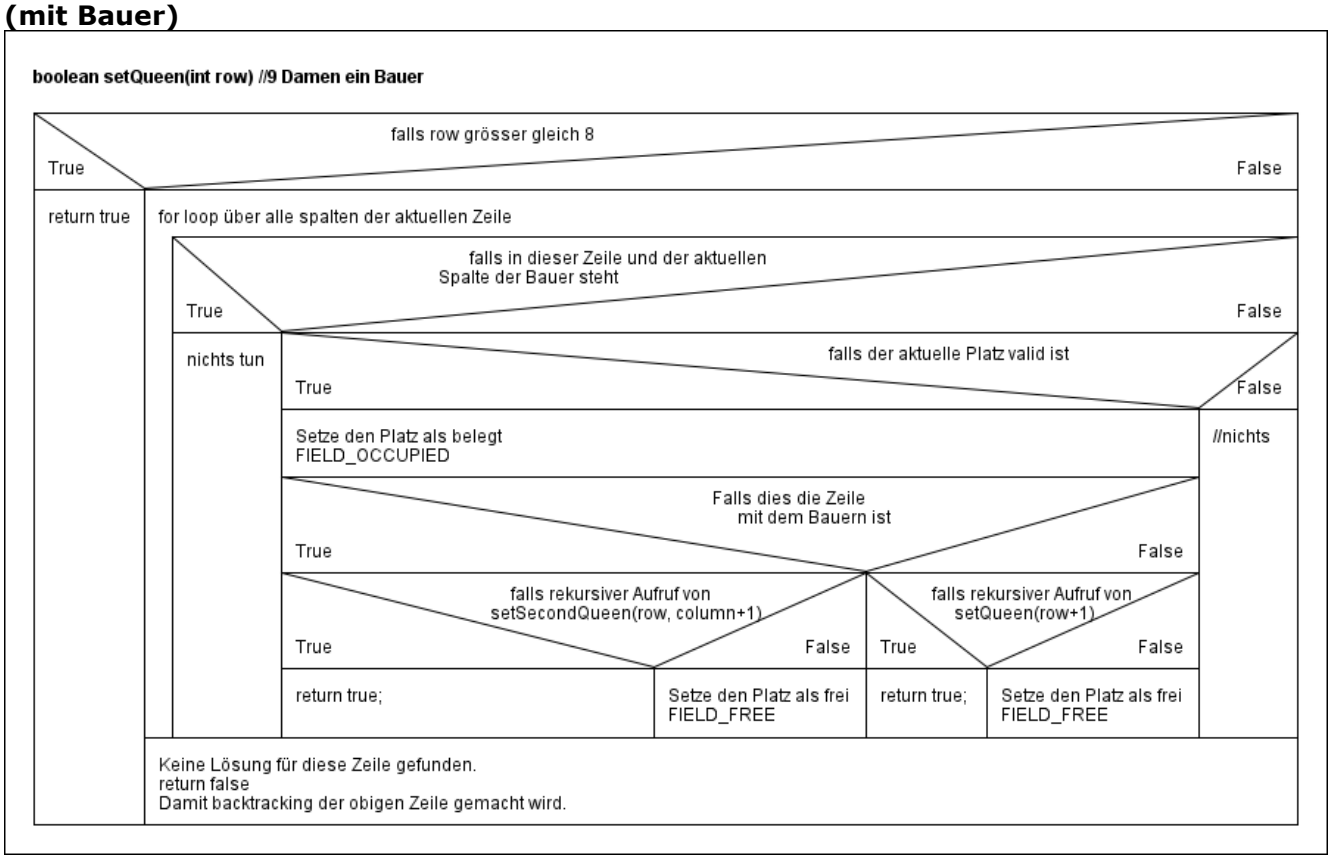
Damen-Problem

Lösung 9 Damen ein Bauer

(ohne Bauer)



(mit Bauer)



Auszug aus Application.java, die Teile die zentral sind

```
14
15          System.out.println("Damen Problem: 9 Damen ein Bauer");
16
17          System.out.println("zusaetzlich kann ein Bauer\nplaziert werden.");
18          if (solver.setPawn(5, 2) == false) {
19              return;
20          }
21
22          int firstSpalte = 2;
23          int firstZeile = 0;
24
25          //Erste Queen setzen
26          solver.setFirstQueen(firstZeile, firstSpalte);
27
28          //Start mit firstZeile + 1 für weiter Queens
29          if (solver.setQueen(firstZeile+1) == false) {
30              System.out.println("Nichts gefunden");
31          }
32          System.out.println("Backtracking-Count: " + solver.getBacktrackingCount());
33
34          System.out.println();
35          //Printout des Spielfeldes
36          for (int i = 0; i < size; i++)
```

DameProblem.java

```java
package ch.bbw.pr.dame;

/**
 * Dame Data-Class
 *
 * @author Peter Rutschmann
 * @version 13.05.2020
 */
public class DameProblem {
    private static final int FIELD_FREE = 0;
    private static final int FIELD_OCCUPIED = 1;
    private static final int FIELD_PAWN = 2;

    private int backtrackingCount = 0;

    private int size;
    private int[][] board;
    private int pawRow = 0;

    public int[][] getBoard() {
        return board;
    }

    public int getBacktrackingCount() {
        return backtrackingCount;
    }

    public void setBacktrackingCount(int count) {
        backtrackingCount = count;
        // System.out.println(count);
    }

    public boolean setPawn(int r, int c) {
        // Der Bauer kann nicht am Rand sitzen !!
        if ((r <= 0) || (c <= 0) || (r >= size-1) || (c > size-1)) {
            System.out.println("Bauer muss im Feld sein und darf nicht am Rand sein. " + r + " " + c);
            return false;
        }
        System.out.println("Bauer ist auf. " + r + " " + c);
        board[r][c] = FIELD_PAWN;
        pawRow = r;
        return true;
    }

    public DameProblem(int size) {
        super();
        this.size = size;
        this.board = new int[size][size];
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                board[i][j] = FIELD_FREE;
            }
        }
    }
```

```
55
56      public void setFirstQueen(int row, int column) {
57          board[row][column] = FIELD_OCCUPIED;
58      }
59
60      public boolean setQueen(int row) {
61          if (row >= size) {
62              return true;
63          }
64
65          // In der aktuellen Zeile alle Spalten nach Möglichkeiten untersuchen
66          for (int column = 0; column < size; column++) {
67
68              // falls da schon der Bauer sitzt
69              if (board[row][column] == FIELD_PAWN) {
70                  // nichts tun try next column
71
72                  // Aktuelle Spalte prüfen
73              } else if (isValid(row, column)) {
74                  // Spalte ist ok, Feld besetzen
75                  board[row][column] = FIELD_OCCUPIED;
76
77                  /*
78                   * Spezialfall, ich muss einen weitere Dame in dieser Spalte probieren
79                   */
80                  if (row == pawRow) {
81
82                      if (setSecondQueen(row, column + 1)) {
83                          // Hat funktioniert mit zweiter Dame und alle folge Zeilen
84                          return true;
85
86                      } else {
87                          /*
88                           * Das ging nicht mit der Dame auf der gleichen Zeile Backtracking machen, also
89                           * Feld wieder Freigeben und nächste Spalte in der Zeile ausprobieren.
90                           */
91                          if (board[row][column] == FIELD_PAWN) {
92                              System.out.println("Bauer!! D");
93                          }
94                          board[row][column] = FIELD_FREE;
95                          setBacktrackingCount(backtrackingCount + 1);
96                      }
97                  } else {
98
99                      /*
100                      * Nächste Queen eine Zeile tiefer ausprobieren -> wenn das erfolgreich ist ok
101                      * -> return true
102                      */
103                     if (setQueen(row + 1)) {
104                         return true;
105                     } else {
106                         /*
107                          * Das ging nicht mit der Dame auf der nächsten Zeile Backtracking machen, also
108                          * Feld wieder Freigeben und nächste Spalte in der Zeile ausprobieren.
109                          */
```

```
110                         if (board[row][column] == FIELD_PAWN) {
111                             System.out.println("Bauer!! C");
112                         }
113                         board[row][column] = FIELD_FREE;
114                         setBacktrackingCount(backtrackingCount + 1);
115                     }
116                 }
117             }
118 
119         }
120         return false;
121     }
122 
123     /*
124      * zweite Dame in der gleichen Spalte probieren
125      */
126     public boolean setSecondQueen(int row, int col) {
127         // Keine Spalte gefunden
128         if (col >= size) {
129             return false;
130         }
131 
132         /*
133          * In der aktuellen Zeile alle Spalten nach Möglichkeiten untersuchen ab der
134          * Spalte aus dem Parameter
135          */
136         for (int column = col; column < size; column++) {
137             // Spalte ist ok, Feld besetzen
138 
139             // falls da schon der Bauer sitzt..
140             if (board[row][column] == FIELD_PAWN) {
141                 // mach nichts
142 
143                 // Aktuelle Spalte prüfen
144             } else if (isValid(row, column)) {
145                 // Spalte ist ok, Feld besetzen
146                 board[row][column] = FIELD_OCCUPIED;
147 
148                 /*
149                  * Nächste Queen eine Zeile tiefer ausprobieren -> wenn das erfolgreich ist ok
150                  * -> return true
151                  */
152                 if (setQueen(row + 1)) {
153                     return true;
154                 } else {
155                     /*
156                      * Das ging nicht mit der Dame auf der nächsten Zeile Backtracking machen, also
157                      * Feld wieder Freigeben und nächste Spalte in der Kolonne ausprobieren.
158                      */
159                     if (board[row][column] == FIELD_PAWN) {
160                         System.out.println("Bauer!! B");
161                     }
162                     board[row][column] = FIELD_FREE;
163                     setBacktrackingCount(backtrackingCount + 1);
164                 }
```

```
165                 }
166             }
167             return false;
168         }
169 
170         private boolean isValid(int r, int c) {
171             int i, j;
172             /*
173              * Suche nach links ob es eine Dame hat -> die erste in der Zeile Falls zwei
174              * Damen in der Zeile sein sollen
175              */
176             for (i = c; i >= 0; i--) {
177                 if (board[r][i] == FIELD_PAWN) {
178                     // System.out.println("Pawn break 1: " + r);
179                     break;
180                 } else if (board[r][i] == FIELD_OCCUPIED) {
181                     return false;
182                 }
183             }
184 
185             /*
186              * Suche in der gleichen Spalte oberhalb ob es eine Dame hat
187              */
188             for (i = r; i >= 0; i--) {
189                 if (board[i][c] == FIELD_PAWN) {
190                     // System.out.println("Pawn break 1: " + r);
191                     break;
192                 } else if (board[i][c] == FIELD_OCCUPIED) {
193                     return false;
194                 }
195             }
196             // Suche Diagonal nach links oben
197             i = r - 1;
198             j = c - 1;
199             while ((i >= 0) && (j >= 0)) {
200                 if (board[i][j] == FIELD_PAWN) {
201                     // System.out.println("Pawn break 2: " + r);
202                     break;
203                 } else if (board[i][j] == FIELD_OCCUPIED) {
204                     return false;
205                 }
206                 i--;
207                 j--;
208             }
209             // Suche nach rechts oben
210             i = r - 1;
211             j = c + 1;
212             while ((i >= 0) && (j < size)) {
213                 if (board[i][j] == FIELD_PAWN) {
214                     // System.out.println("Pawn break 3: " + r);
215                     break;
216                 } else if (board[i][j] == FIELD_OCCUPIED) {
217                     return false;
218                 }
219                 i--;
220                 j++;
221             }
222             return true;
223         }
224 }
```