Übungen zum Vortrag „Backtracking mit Heuristiken“

A) Java-Implementierung studieren von „Backtracking im Labyrinth“

B) Pseudocode schreiben zu „Backtracking beim n Damen Problem“

C) Implementierung der Springerwege (optional)

A) Java-Implementierung studieren von „Backtracking im Labyrinth“

Vorgehen:

1. Handouts zu den Vortragsfolien und Sourcecode zum Programm „Labyrinth“ verteilen.

2. Die Schüler studieren den Pseudocode zu „Backtracking im Labyrinth“ im Handout und vergleichen mit der Java Implementierung FindeLoesung aus Labyrinth.java.

3. Besprechung der Java Implementierung anhand der folgenden Folien.

A) Java-Implementierung „Labyrinth“

Zuerst definieren wir eine **Datenstruktur**, welche die **Lösung** des Labyrinths enthalten wird. Als Lösung werden die Labyrinthfelder auf dem gefundenen Weg in der Besuchsreihenfolge durchnummeriert.

```
public class Lsg {
    int[][] feld;    // Labyrinthfelder im 2D-Koord.system

    public Lsg(int K, int L) { // Konstruktor
        feld = new int[K][L]; // Labyrinth mit KxL Feldern
    }
}
```

Aufruf von **FindeLoesung**:

```
// Zufälliges Labyrinth erzeugen,
// Start (startX, startY) und Ziel (zielX, zielY) definieren
// und Datenstruktur loesung für Lösung mit new erzeugen
newLabyrinth(12,12);
// Startfeld markieren
loesung.feld[startX][startY] = 1;

// FindeLoesung(int index, Lsg loesung, int aktX, int aktY)
if (FindeLoesung(2, loesung, startX, startY)) {
    // Die gefundene Lösung steht in loesung
    System.out.println( „Loesung gefunden! “);
} else
    System.out.println( „Keine Loesung! “);
```

Java-**Implementierung** von **FindeLoesung:**

```
final static int LEER = 0, SACKGASSE = -1;   // Feldinformation
final static int[] STEPX = {  0, 1, 0,-1 };      // STEPX,-Y: Schritte
final static int[] STEPY = { -1, 0, 1, 0 };      // in alle Richtungen

boolean FindeLoesung(int index, Lsg loesung, int aktX, int aktY) {
    // index          = aktuelle Schrittzahl
    // loesung        = Referenz auf bisherige Teil-Lösung
    // aktX, aktY     = aktuelle Feldposition im Labyrinth
    int schritt = -1;

    // while(es gibt es noch neue Teil-Lösungsschritte)
    while (schritt != LINKS) {
        // Wähle einen neuen Teil-Lösungsschritt schritt;
        schritt ++; // Weg nach 1. oben, 2. rechts, 3. unten
                    // und 4. links zu erweitern versuchen.
        int neuX    = aktX + STEPX[schritt];
        int neuY    = aktY + STEPY[schritt];
```

A) Java-Implementierung „Labyrinth“

```
// Tests, ob schritt gültig ist

boolean ok = true;
// Test, ob schritt innerhalb Brett bleibt
if (neuX < 0 || neuX >= K) ok = false;
if (neuY < 0 || neuY >= L) ok = false;
// Test, ob schritt durch Wand führt (sofern innerhalb)
if (ok && hatWand(aktX,aktY,neuX,neuY,schritt))
    ok = false;
// Test, ob schritt auf ein bereits besuchtes Feld führt
if (ok && loesung.feld[neuX][neuY] != LEER)
    ok = false;

// if (schritt ist gültig)
if (ok) {
    // Erweitere loesung um schritt
    // Markiere neues Feld mit aktueller Schrittzahl
    loesung.feld[neuX][neuY] = index;
```

A) Java-Implementierung „Labyrinth“

```
                // if (loesung noch nicht vollständig)
                if (!ausgangGefunden(neuX, neuY) {
                    // rekursiver Aufruf von FindeLoesung
                    if (FindeLoesung(index+1, loesung, neuX, neuY)) {
                        // Lösung gefunden
                        return true;
                    } else {
                        // Wir sind in einer Sackgasse:
                        // Mache schritt rückgängig: Backtracking
                        loesung.feld[neuX][neuY] = SACKGASSE;
                    }
                } else return true; // Lösung gefunden -> fertig
            }
        }
        return false; // keine Lösung gefunden
}
// Bei true als Rückgabewert steht die Lösung in loesung
```

B) Pseudocode schreiben zu „Backtracking beim n Damen Problem “

Vorgehen:

1. Aufgabenstellung (gemäss folgender zwei Folien) vorstellen.

2. Die Schüler erarbeiten selbständig einen Lösungs-vorschlag zum n Damenproblem als Pseudocode. Sie können dabei auf das Handout und die Lösung zum Labyrinth zurückgreifen.

3. Lösung als Pseudocode vorstellen (gemäss Folien).

4. Demontration des animierten Algorithmus mit dem Javaprogramm „Dame “.

Aufgabe:
Stellen Sie n Damen auf einem nxn Schachbrett so auf, dass keine die andere bedroht.

Lösen Sie diese Aufgabe *auf Papier in Java und Pseudocode* mit einem Backtracking Algorithmus für 3 <= n <= 14. Nehmen Sie den Code zum allgemeinen Backtracking Algorithmus als Ausgangslage. Sie haben *20 Minuten* Zeit.

Details:
Eine Dame bedroht gegnerische Schachfiguren, die auf der gleichen Zeile, Spalte oder Diagonale stehen wie sie.

Die Prozedur soll folgende Signatur haben:

boolean **FindeLoesung**(int spalte, **DameLsg** loesung)

Sie wird als FindeLoesung(1, loesung) aufgerufen. Das Resultat steht bei Rückgabe von true in loesung.

Hinweise zur Implementierung:

Für eine effiziente Implementierung soll die Stellung der Damen in folgenden Variablen gehalten werden:

```
boolean z[]    = new boolean[n];     // besetzte Zeile
boolean d1[]   = new boolean[2n];    // besetzte / Diagonalen
boolean d2[]   = new boolean[2n-1]; // besetzte \ Diagonalen
int dame[]     = new int[n];
      // i-te Dame steht in Spalte i und Zeile dame[i]
```

Diese Variablen sind in der Klasse **DameLsg** enthalten und mit loesung.z, loesung.d1 usw. ansprechbar. Sie verhindern, dass das Schachbrett zeilen-, spalten- und diagonalenweise nach Bedrohungen abgesucht werden muss.

Bem. Zeilen- und Spaltennummern beginnen hier mit 0.

Lösung als Pseudocode:

```
boolean FindeLoesung(int spalte, DameLsg loesung) {
    // spalte  = aktuelle Schrittzahl und zugleich aktuelle Spalte
    // loesung = Referenz auf bisherige Teil-Lösung
    int zeile = -1;

    // while(es gibt es noch neue Teil-Lösungsschritte)
    while (zeile != n-1) {
        // Wähle einen neuen Teil-Lösungsschritt schritt;
        zeile++; // Zeilen von oben nach unten ausprobieren

        // if (schritt ist gültig),
        if („Dame (zeile, spalte) wird nicht bedroht“) {
            // Erweitere loesung um schritt
            „Platziere Dame auf (zeile, spalte) in dame[] und führe
            die neu ausgeübten Bedrohungen in z, d1, d2 nach. “
```

Lösung als Pseudocode (Fortsetzung):

```
            // if (loesung noch nicht vollständig)
            if (spalte != n-1) {
                // rekursiver Aufruf von FindeLoesung
                if (FindeLoesung(spalte+1, loesung)) {
                    // Lösung gefunden
                    return true;
                } else {
                    // Wir sind in einer Sackgasse:
                    „Mache schritt rückgängig mit Backtracking:
                    Lösche Dame und ihre ausgeübten Bedrohungen.“
                }
            } else return true; // Lösung gefunden -> fertig
        }
    } return false; // keine Lösung gefunden
}
```

C) Implementierung der Springerwege (optional)

Aufgabe:
Als grössere optionale Übung implementieren die Schüler den im Vortrag vorgestellten Backtracking Algorithmus zu den Springerwegen inklusive der Heuristik von Warnsdorf.

Damit sollen dann mögliche Springerwege für verschiedene Schachbrettgrössen berechnet werden.

Zusatz:

- Springer*kreise* finden (Startfeld = Endfeld)
- Erklären, warum es bei nxn mit n ungerade keinen Springerkreis geben kann