

# Alternative 7 - Labyrinth

**Präsentation zum Code**

**Betriebssysteme & Rechnernetze**

# Agenda

1. Aufgabenstellung
2. Benutzereingabe
3. Initiierung des Arrays
4. Rekursives Backtracking
5. Ausgabe
6. Spielerbewegung

# 1. Aufgabenstellung

- Aufgabe: “Perfect Maze”
- Anforderungen:
  - 1 Eingang – 1 Ausgang – 1 Pfad
  - Manuelle Benutzereingabe zur Bestimmung der Größe
  - Zufällige Generierung eines Labyrinths
  - Spielerbewegung durch Tastatureingaben
  - Nur Schritte in freie Felder zulässig
  - Gewinnnachricht bei Erreichung des Ziels

## 2. Benutzereingabe

```
1 function input_achsen (  
2     local sAchse=$1  
3     while true; do  
4         read -p "$sAchse: " nVarAchse  
5         if [[ ! $nVarAchse =~ ^[0-9]+$ ]];then  
6             echo "Bitte nur Zahlen eingeben!"  
7         elif [[ $nVarAchse -gt 29 ]] || [[ $nVarAchse -lt 5 ]];then  
8             echo "Bitte zwischen 5 und 29 eingeben!"  
9         elif [[ $(($nVarAchse % 2)) -eq 0 ]];then  
10            echo "Bitte eine ungerade Zahl eingeben!"  
11        else  
12            if [[ $sAchse == "Höhe" ]];then  
13                nHoehe=$((nVarAchse + 2))  
14            else  
15                nBreite=$((nVarAchse + 2))  
16            fi  
17            break  
18        fi  
19    done  
20 }
```

- Interaktive Benutzereingabe
- Funktion: "input\_achsen"
- Prüfung:
  - Nur Zahlen möglich
  - Zahlen zwischen 5 und 29
  - Ungerade Zahlen

## 3. Initiierung des Arrays

```
1 function lab_array_initiieren {
2   for ((y=0; y<nHoehe; y++)) ; do
3     for ((x=1; x<$(nBreite-1); x++)) ; do
4       lab_array[$((y * nBreite + x))]=0
5     done
6     lab_array[$((y * nBreite + 0))]=1
7     lab_array[$((y * nBreite + (nBreite - 1)))]=1
8   done
9   for ((x=0; x<nBreite; x++)) ; do
10    lab_array[$x]=1
11    lab_array[$((nHoehe - 1) * nBreite + x)]=1
12  done
13  lab_array[$((nBreite + 2))]=2
14  lab_array[$((nHoehe - 2) * nBreite + nBreite - 3)]=1
15 }
```

- Funktion “lab\_array\_initiieren”
- Möglichen Werte:
  - 0 = eine Wand
  - 1 = leeres, begehbare Feld
  - 2 = Spieler

## 3. Initiierung des Arrays

1	1	1	1	1	1	1
1	0	2	0	0	0	1
1	0	0	0	0	0	1
1	0	0	0	0	0	1
1	0	0	0	0	0	1
1	0	0	0	1	0	1
1	1	1	1	1	1	1

Tabelle 1: Array Ausgabe

	■	•	■	■	■	
	■	■	■	■	■	
	■	■	■	■	■	
	■	■	■	■	■	
	■	■	■		■	

Tabelle 2: Ausgabe mit Symbolen

- Höhe/Breite von 7 = sichtbares Feld auf 5x5
- Labyrinth von leeren Feldern umgeben
- Benutzereingabe in "input\_achsen" müssen um 2 erhöht werden

# 4. Rekursives Backtracking – Startzelle bestimmen

```

1 rekursives_backtracking $((2 * nBreite + 2))

1 function rekursives_backtracking {
2   local nAktuelle_zelle=$1
3   local nZufallszahl=$RANDOM
4   local i=0
5   lab_array[$nAktuelle_zelle]=1
6   while [ $i -le 3 ] ; do
7     nModZufallszahl=$((nZufallszahl % 4))
8     if [[ $nModZufallszahl -eq 0 ]];then
9       nRichtung=1
10    elif [[ $nModZufallszahl -eq 1 ]];then
11      nRichtung=-1
12    elif [[ $nModZufallszahl -eq 2 ]];then
13      nRichtung=$nBreite
14    elif [[ $nModZufallszahl -eq 3 ]];then
15      nRichtung=$((-$nBreite))
16    fi
17    local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18    if [[ lab_array[$nNaechste_zelle] -eq 0 ]];then
19      local nUebernaechste_zelle=$((nNaechste_zelle + nRichtung))
20      if [[ lab_array[$nUebernaechste_zelle] -eq 0 ]];then
21        lab_array[$nNaechste_zelle]=1
22        rekursives_backtracking $nUebernaechste_zelle
23      fi
24    fi
25    i=$((i + 1))
26    nZufallszahl=$((nZufallszahl + 1))
27  done
28 }

```

	■	•	■	■	■	
	■	1	■	■	■	
	■	■	■	■	■	
	■	■	■	■	■	
	■	■	■		■	

Tabelle 3: Labyrinth nach rekursivem Backtracking

# 4. Rekursives Backtracking – Nachbarzelle bestimmen

```

1 rekursives_backtracking $((2 * nBreite + 2))

1 function rekursives_backtracking {
2   local nAktuelle_zelle=$1
3   local nZufallszahl=$RANDOM
4   local i=0
5   lab_array[$nAktuelle_zelle]=1
6   while [ $i -le 3 ]; do
7     nModZufallszahl=$((nZufallszahl % 4))
8     if [[ $nModZufallszahl -eq 0 ]];then
9       nRichtung=1
10    elif [[ $nModZufallszahl -eq 1 ]];then
11      nRichtung=-1
12    elif [[ $nModZufallszahl -eq 2 ]];then
13      nRichtung=$nBreite
14    elif [[ $nModZufallszahl -eq 3 ]];then
15      nRichtung=$((-$nBreite))
16    fi
17    local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18    if [[ lab_array[$nNaechste_zelle] -eq 0 ]];then
19      local nUebernaechste_zelle=$((nNaechste_zelle + nRichtung))
20      if [[ lab_array[$nUebernaechste_zelle] -eq 0 ]];then
21        lab_array[$nNaechste_zelle]=1
22        rekursives_backtracking $nUebernaechste_zelle
23      fi
24    fi
25    i=$((i + 1))
26    nZufallszahl=$((nZufallszahl + 1))
27  done
28 }

```

	■	•	■	■	■	
	■	1	■	■	■	
	■	■	■	■	■	
	■	■	■	■	■	
	■	■	■		■	

Tabelle 3: Labyrinth nach rekursivem Backtracking



# 4. Rekursives Backtracking – Rekursion

```

1 rekursives_backtracking $((2 * nBreite + 2))

1 function rekursives_backtracking {
2     local nAktuelle_zelle=$1
3     local nZufallszahl=$RANDOM
4     local i=0
5     lab_array[$nAktuelle_zelle]=1
6     while [ $i -le 3 ] ; do
7         nModZufallszahl=$((nZufallszahl % 4))
8         if [[ $nModZufallszahl -eq 0 ]];then
9             nRichtung=1
10        elif [[ $nModZufallszahl -eq 1 ]];then
11            nRichtung=-1
12        elif [[ $nModZufallszahl -eq 2 ]];then
13            nRichtung=$nBreite
14        elif [[ $nModZufallszahl -eq 3 ]];then
15            nRichtung=$((-$nBreite))
16        fi
17        local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18        if [[ lab_array[$nNaechste_zelle] -eq 0 ]];then
19            local nUebernaechste_zelle=$((nNaechste_zelle + nRichtung))
20            if [[ lab_array[$nUebernaechste_zelle] -eq 0 ]];then
21                lab_array[$nNaechste_zelle]=1
22                rekursives_backtracking $nUebernaechste_zelle
23            fi
24        fi
25        i=$((i + 1))
26        nZufallszahl=$((nZufallszahl + 1))
27    done
28 }

```

	■	•	■	■	■	
	■	1		2	■	
	■	■	■	■	■	
	■	■	■	■	■	
	■	■	■		■	

Tabelle 3: Labyrinth nach rekursivem Backtracking

# 4. Rekursives Backtracking – Backtracking

```

1 rekursives_backtracking $((2 * nBreite + 2))

1 function rekursives_backtracking {
2   local nAktuelle_zelle=$1
3   local nZufallszahl=$RANDOM
4   local i=0
5   lab_array[$nAktuelle_zelle]=1
6   while [ $i -le 3 ] ; do
7     nModZufallszahl=$((nZufallszahl % 4))
8     if [[ $nModZufallszahl -eq 0 ]];then
9       nRichtung=1
10    elif [[ $nModZufallszahl -eq 1 ]];then
11      nRichtung=-1
12    elif [[ $nModZufallszahl -eq 2 ]];then
13      nRichtung=$nBreite
14    elif [[ $nModZufallszahl -eq 3 ]];then
15      nRichtung=$((-$nBreite))
16    fi
17    local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18    if [[ lab_array[$nNaechste_zelle] -eq 0 ]];then
19      local nUebernaechste_zelle=$((nNaechste_zelle + nRichtung))
20      if [[ lab_array[$nUebernaechste_zelle] -eq 0 ]];then
21        lab_array[$nNaechste_zelle]=1
22        rekursives_backtracking $nUebernaechste_zelle
23      fi
24    fi
25    i=$((i + 1))
26    nZufallszahl=$((nZufallszahl + 1))
27  done
28 }

```

	■	•	■	■	■	
	■	1		2	■	
	■	■	■		■	
	■	4		3	■	
	■	■	■		■	

Tabelle 3: Labyrinth nach rekursivem Backtracking

## 4. Rekursives Backtracking – Besonderheiten

### Leere Felder außerhalb des Labyrinths

	■	•	■	■	■	
	■	1		2	■	
	■	■	■		■	
	■	4		3	■	
	■	■	■		■	

Tabelle 3: Labyrinth nach rekursivem Backtracking

### Gerade Zahl bei Breite

	■	•	■	■	
	■	1		2	
	■	■	■		
	■	4		3	
	■	■	■		

Tabelle 3: Labyrinth nach rekursivem Backtracking

## 5. Ausgabe

```
1 function labyrinth_ausgeben {
2   for ((y=0; y<nHoehe; y++)) ; do
3     for ((x = 0; x<nBreite; x++ )) ; do
4       if [[ lab_array[$((y * nBreite + x))] -eq 0 ]];then
5         echo -n -e "\u2588\u2588"
6       elif [[ lab_array[$((y * nBreite + x))] -eq 2 ]];then
7         echo -n -e "\u25CF "
8       else
9         echo -n " "
10      fi
11    done
12    echo
13  done
14 }
```

- Funktion: "labyrinth\_ausgeben"
- UTF-8-Symbol
- Wert 0 = Wand = "■"
- Wert 2 = Spieler = "●"
- Andernfalls leeres Feld
- Zeilenumbruch durch leeres „echo“

## 6. Spielerbewegung

```

1 while [ $sRichtung != "q" ];do
2   read -nl -s sRichtung
3   clear
4   labyrinth_ausgeben
5
6   if [[ $sRichtung == a ]];then
7     nNeue_position=$((nAktuelle_position - 1))
8     kollisionspruefung
9   elif [[ $sRichtung == d ]];then
10    nNeue_position=$((nAktuelle_position + 1))
11    kollisionspruefung
12   elif [[ $sRichtung == s ]];then
13    nNeue_position=$((nAktuelle_position + nBreite))
14    kollisionspruefung
15   elif [[ $sRichtung == w ]];then
16    nNeue_position=$((nAktuelle_position - nBreite))
17    if [[ $nNeue_position -lt $nEingang ]];then
18      echo "Falsche Richtung. Der Ausgang befindet sich unten rechts."
19      echo -e "Züge = $nZuege\n"
20      echo -e "Bewegung mit den Pfeiltasten: asdw\nSpiel Beenden: q"
21      continue
22    fi
23    kollisionspruefung
24  fi

```

```

26   if [[ $nAktuelle_position -gt $nAusgang ]];then
27     echo "Züge = $nZuege"
28     echo "Herzlichen Glückwunsch! Du hast $nZuege Züge gebraucht."
29     read
30     exit
31   fi
32
33   if [[ ! $sRichtung == *[asdwq] ]];then
34     echo "Bitte nur die Buchstaben asdw oder q eingeben!"
35   fi
36
37   echo -e "Züge = $nZuege\n"
38   echo -e $sEingaben
39 done
40
41 clear
42 labyrinth_ausgeben
43 echo -e "Züge = $nZuege\n"
44 read -p "Das Spiel wurde beendet."

```

## 6. Spielerbewegung - Benutzereingabe

```
1 while [ $sRichtung != "q" ];do
2   read -nl -s sRichtung
3   clear
4   labyrinth_ausgeben
5
6   if [[ $sRichtung == a ]];then
7     nNeue_position=$((nAktuelle_position - 1))
8     kollisionspruefung
9   elif [[ $sRichtung == d ]];then
10    nNeue_position=$((nAktuelle_position + 1))
11    kollisionspruefung
12  elif [[ $sRichtung == s ]];then
13    nNeue_position=$((nAktuelle_position + nBreite))
14    kollisionspruefung
15  elif [[ $sRichtung == w ]];then
16    nNeue_position=$((nAktuelle_position - nBreite))
```

- Einlesen der Richtungstaste
- Prüfung der Eingabe
- Berechnung der Variable „nNeue\_Position“
- Aufruf Funktion „kollisionsprüfung“

## 6. Spielerbewegung - Kollisionsprüfung

```
1 function kollisionspruefung {  
2   if [[ ${lab_array[$nNeue_position]} = 1 ]];then  
3     lab_array[$nAktuelle_position]=1  
4     lab_array[$nNeue_position]=2  
5     nAktuelle_position=$nNeue_position  
6     let nZuege++  
7     clear  
8     labyrinth_ausgeben  
9   else  
10    echo "Hier ist eine Mauer."  
11  fi  
12 }
```

- Prüfung, ob in der nächsten Zelle eine leere Zelle ist
- Spieler wird in nächste Zelle bewegt
- Ansonsten keine Bewegung
- Ausgabe „Hier ist eine Mauer.“

## 6. Spielerbewegung - Sonderfälle

### 1. Eingangsprüfung

```
15 elif [[ $sRichtung == w ]];then
16     nNeue_position=$((nAktuelle_position - nBreite))
17     if [[ $nNeue_position -lt $nEingang ]];then
18         echo "Falsche Richtung. Der Ausgang befindet sich unten rechts."
19         echo -e "Züge = $nZuege\n"
20         echo -e "Bewegung mit den Pfeiltasten: asdw\nSpiel Beenden: q"
```

### 2. Eingabeprüfung

```
33 if [[ ! $sRichtung == *[asdwq] ]];then
34     echo "Bitte nur die Buchstaben asdw oder q eingeben!"
35 fi
```

### 3. Abbruch mit "q"

```
41 clear
42 labyrinth_ausgeben
43 echo -e "Züge = $nZuege\n"
44 read -p "Das Spiel wurde beendet."
```

#### Prüfung der Sonderfälle

1. Prüfung, ob sich der Spieler über die Zelle des Eingangs bewegen will
2. Prüfung, ob die Eingabe nur aus den gültigen Buchstaben besteht
3. Ausgabe, wenn die while-Schleife mit q abgebrochen wird



## 6. Spielerbewegung - Gewinnkondition

```
26  if [[ $nAktuelle_position -gt $nAusgang ]];then
27      echo "Züge = $nZuege"
28      echo "Herzlichen Glückwunsch! Du hast $nZuege Züge gebraucht."
29      read
30      exit
31  fi
```

- Prüfung, ob Spieler sich unterhalb des Ausgangs bewegt hat
- Wenn ja, ist das Labyrinth erfolgreich durchlaufen
- Ausgabe Anzahl benötigter Züge & Gewinnernachricht
- Spiel wird beendet

# Vielen Dank für Ihre Aufmerksamkeit

## 4. Rekursives Backtracking - Funktionsweise

- Schritte des Algorithmus:
  1. Wähle zufällige Zelle aus und betrachte diese als aktuelle Zelle
  2. Markiere diese Zelle als „besucht“
  3. Solange diese Zelle „unbesuchte“ Nachbarzellen hat
    - Wähle eine dieser Nachbarzellen zufällig aus
    - Entferne die Wand zwischen den beiden Zellen
    - Ausgewählte Nachbarzelle = aktuelle Zelle
    - Wiederhole die Schritte rekursiv