

# Alternative 7 - Labyrinth

**Präsentation zum Code** 

Betriebssysteme & Rechnernetze



## Agenda

- Aufgabenstellung
- Benutzereingabe
- Initiierung des Arrays
- 4. Rekursives Backtracking
- 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

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

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



## 3. Initiierung des Arrays

```
function lab array initiieren
      for ((y=0; y<nHoehe; y++)); do
        for ((x=1; x<$((nBreite-1)); x++)); do
          lab array[$((y * nBreite + x))]=0
        done
        lab array[$((y * nBreite + 0))]=1
        lab array[$((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, begehbares 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

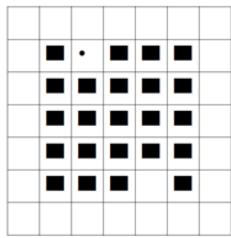


Tabelle 2: Ausgabe mit Symbolen

- 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

```
rekursives backtracking $((2 * nBreite + 2))
    function rekursives backtracking (
      local naktuelle zelle=01
      local nZufallszahl-$RANDOM
      local i=0
      lab array[$nAktuelle zelle]=1
        while [ $1 -le 3 ] ; do
          nModZufallszahl=$((nZufallszahl % 4))
          if [[ SnModZufallszahl -eq 0 ]]; then
            nRichtung=1
          elif [[ SnModZufallszahl -eq 1 ]]; then
            nRichtung=-1
12
          elif [[ SnModZufallszahl -eq 2 ]];then
13
            nRichtung=$nBreite
14
          elif [[ SnModZufallszahl -eq 3 ]];then
15
            nRichtung=$((-$nBreite))
16
          fi
17
          local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18
            if [[ lab array[$nNaechste zelle] -eq 0 ]];then
19
              local nUebernaechste zelle=$((nNaechste zelle + nRichtung))
20
              if [[ lab array[SnUebernaechste zelle] -eq 0 ]]; then
                lab array[SnNaechste zelle]=1
                rekursives backtracking SnUebernaechste zelle
23
              21
24
            25
25
          i=$((i + 1))
26
          nZufallszahl=$((nZufallszahl + 1))
        done
```

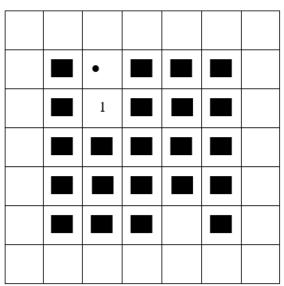


Tabelle 3: Labyrinth nach rekursivem Backtracking



#### 4. Rekursives Backtracking – Nachbarzelle bestimmen

```
rekursives backtracking $((2 * nBreite + 2))
    function rekursives_backtracking {
      local nAktuelle zelle=01
      local nZufallszahl-$RANDOM
      local 1=0
      lab array[$nAktuelle zelle]=1
        while [ $1 -le 3 ] ; do
          nModZufallszahl=$((nZufallszahl % 4))
          if [[ SnModZufallszahl -eq 0 ]]; then
            nRichtung=1
          elif [[ SnModZufallszahl -eq 1 ]]; then
            nRichtung=-1
          elif [[ SnModZufallszahl -eq 2 ]];then
            nRichtung=$nBreite
          elif [[ SnModZufallszahl -eq 3 ]];then
            nRichtung=$((-$nBreite))
16
          fi
          local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
            if [[ lab array[$nNaechste zelle] -eq 0 ]]; then
              local nUebernaechste zelle=$((nNaechste zelle + nRichtung))
              if [[ lab array[SnUebernaechste zelle] -eq 0 ]]; then
21
                lab array[SnNaechste zelle]=1
                rekursives backtracking SnUebernaechste zelle
23
              21
24
            25
25
          i=$((i + 1))
26
          nZufallszahl=$((nZufallszahl + 1))
        done
```

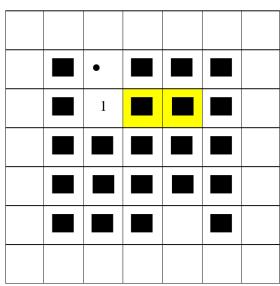


Tabelle 3: Labyrinth nach rekursivem Backtracking



#### 4. Rekursives Backtracking – Rekursion

```
rekursives backtracking $((2 * nBreite + 2))
    function rekursives backtracking (
      local naktuelle zelle-Sl
      local nZufallszahl-$RANDOM
      local i=0
      lab array[$nAktuelle zelle]=1
        while [ $1 -le 3 ] ; do
          nModZufallszahl=$((nZufallszahl % 4))
          if [[ SnModZufallszahl -eq 0 ]]; then
            nRichtung=1
          elif [[ SnModZufallszahl -eq 1 ]]; then
            nRichtung=-1
12
          elif [[ SnModZufallszahl -eq 2 ]];then
13
            nRichtung=$nBreite
14
          elif [[ SnModZufallszahl -eq 3 ]];then
15
            nRichtung=$((-$nBreite))
16
          fi
17
          local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18
            if [[ lab array[$nNaechste zelle] -eq 0 ]];then
19
              local nUebernaechste zelle=$((nNaechste zelle + nRichtung))
              if [[ lab array[SnUebernaechste zelle] -eq 0 ]]; then
                lab array[$nNaechste zelle]=1
                rekursives backtracking SnUebernaechste zelle
23
              21
24
            21
25
          i=$((i + 1))
26
          nZufallszahl=$((nZufallszahl + 1))
        done
```

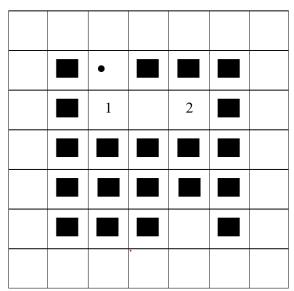


Tabelle 3: Labyrinth nach rekursivem Backtracking



#### 4. Rekursives Backtracking – Backtracking

```
rekursives backtracking $((2 * nBreite + 2))
    function rekursives backtracking (
      local nAktuelle zelle=01
      local nZufallszahl-$RANDOM
      local i=0
      lab array[$nAktuelle zelle]=1
        while [ $1 -le 3 ] ; do
          nModZufallszahl=$((nZufallszahl % 4))
          if [[ SnModZufallszahl -eq 0 ]]; then
            nRichtung=1
          elif [[ SnModZufallszahl -eq 1 ]]; then
            nRichtung=-1
12
          elif [[ SnModZufallszahl -eq 2 ]];then
13
            nRichtung=$nBreite
14
          elif [[ SnModZufallszahl -eq 3 ]];then
15
            nRichtung=$((-$nBreite))
16
          fi
17
          local nNaechste_zelle=$((nAktuelle_zelle + nRichtung))
18
            if [[ lab array[$nNaechste zelle] -eq 0 ]];then
19
              local nUebernaechste zelle=$((nNaechste zelle + nRichtung))
20
              if [[ lab array[SnUebernaechste zelle] -eq 0 ]]; then
21
                lab array[$nNaechste zelle]=1
                rekursives backtracking SnUebernaechste zelle
23
              f1
24
          i=$((i + 1))
          nZufallszahl=S((nZufallszahl + 1))
        done
```

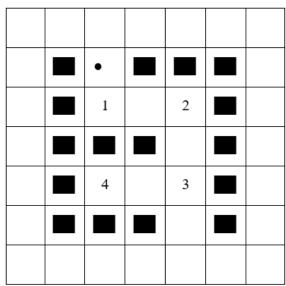


Tabelle 3: Labyrinth nach rekursivem Backtracking



### 4. Rekursives Backtracking – Besonderheiten

#### Leere Felder außerhalb des Labyrinths

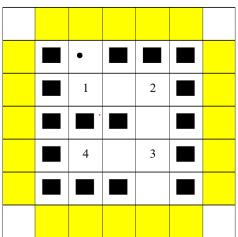


Tabelle 3: Labyrinth nach rekursivem Backtracking

#### Gerade Zahl bei Breite

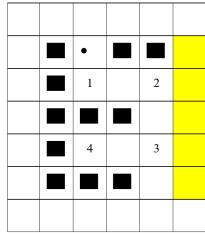


Tabelle 3: Labyrinth nach rekursivem Backtracking



# 5. Ausgabe

```
function labyrinth_ausgeben {
  for ((y=0; y<nHoehe; y++)); do
  for ((x = 0; x<nBreite; x++)); do

  if [[ lab_array[$((y * nBreite + x))] -eq 0 ]]; then
        echo -n -e "\u2588\u2588"
  elif [[ lab_array[$((y * nBreite + x))] -eq 2 ]]; then
        echo -n -e "\u25CF"
        else
        echo -n " "
  fi
  done
  echo
  done
</pre>
```

- Funktion: "labyrinth ausgeben"
- UTF-8-Symbol
- Wert 0 = Wand = "
- Wert 2 = Spieler = "● "
- Andernfalls leeres Feld
- Zeilenumbruch durch leeres "echo"



#### 6. Spielerbewegung

```
while [ $sRichtung != "q" ];do
      read -nl -s sRichtung
      clear
      labyrinth ausgeben
      if [[ $sRichtung == a ]]; then
        nNeue position=$((nAktuelle position - 1))
        kollisionspruefung
      elif [[ $sRichtung == d ]]; then
        nNeue position=$((nAktuelle position + 1))
11
        kollisionspruefung
      elif [[ $sRichtung == s ]]; then
13
        nNeue position=$((nAktuelle position + nBreite))
14
        kollisionspruefung
15
      elif [[ $sRichtung == w ]]; then
16
        nNeue position=$((nAktuelle position - nBreite))
        if [[ $nNeue position -lt $nEingang ]]; then
18
          echo "Falsche Richtung. Der Ausgang befindet sich unten rechts."
19
          echo -e "Züge = $nZuege\n"
          echo -e "Bewegung mit den Pfeiltasten: asdw\nSpiel Beenden: q"
21
          continue
22
        fi
23
        kollisionspruefung
24
```

```
if [[ $nAktuelle position -gt $nAusgang ]]; then
        echo "Züge = $nZuege"
        echo "Herzlichen Glückwunsch! Du hast SnZuege Züge gebraucht."
29
        read
        exit
31
      fi
      if [[ ! $sRichtung == *[asdwq] ]]; then
        echo "Bitte nur die Buchstaben asdw oder q eingeben!"
36
      echo -e "Züge = $nZuege\n"
      echo -e SsEingaben
    done
40
    clear
42 labyrinth ausgeben
43 echo -e "Züge = $nZuege\n"
44 read -p "Das Spiel wurde beendet."
```



## 6. Spielerbewegung - Benutzereingabe

```
while [ $sRichtung != "q" ];do
      read -nl -s sRichtung
      clear
      labyrinth ausgeben
 5
      if [[ $sRichtung == a ]]; then
        nNeue position=$((nAktuelle position - 1))
8
        kollisionspruefung
      elif [[ $sRichtung == d ]];then
        nNeue position=$((nAktuelle position + 1))
10
        kollisionspruefung
11
12
      elif [[ $sRichtung == s ]]; then
13
        nNeue position=$((nAktuelle position + nBreite))
14
        kollisionspruefung
15
      elif [[ $sRichtung == w ]];then
        nNeue position=$((nAktuelle position - nBreite))
```

- Einlesen der Richtungstaste
- Prüfung der Eingabe
- Berechnung der Variable "nNeue Position"
- Aufruf Funktion "kollisionsprüfung"



# 6. Spielerbewegung - Kollisionsprüfung

```
function kollisionspruefung {
   if [[ ${lab_array[$nNeue_position]} = 1 ]];then
    lab_array[$nAktuelle_position]=1
   lab_array[$nNeue_position]=2
   nAktuelle_position=$nNeue_position
   let nZuege++
   clear
   labyrinth_ausgeben
   else
   echo "Hier ist eine Mauer."
   fi
}
```

- 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

```
elif [[ $sRichtung == w ]]; then
nNeue_position=$((nAktuelle_position - nBreite))
if [[ $nNeue_position -1t $nEingang ]]; then
ls echo "Falsche Richtung. Der Ausgang befindet sich unten rechts."
echo -e "Züge = $nZuege\n"
echo -e "Bewegung mit den Pfeiltasten: asdw\nSpiel Beenden: g"
```

#### 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

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



# 6. Spielerbewegung - Gewinnkondition

```
if [[ $nAktuelle_position -gt $nAusgang ]];then
echo "Züge = $nZuege"
echo "Herzlichen Glückwunsch! Du hast $nZuege Züge gebraucht."
read
exit
```

- 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

#### Backup



## 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