

# Jugendwettbewerb Informatik 2019

## Runde 3

### Junioraufgabe 2: Kacheln

25. November 2019

#### Inhaltsverzeichnis

Lösungsidee.....	1
Umsetzung.....	2
Beispiele.....	3
Vorgegebene Beispiele.....	3
Eigene Beispiele.....	5
Quellcode.....	6

#### Lösungsidee

Die Kacheln sind aus 4 Quadraten zusammengesetzt (4-Teilig). Beim Zusammensetzen von Landschaften soll eine Regel beachtet werden. Die Außenseiten der Quadrate dürfen nur so aneinander gelegt werden, dass Land auf Land und Wasser auf Wasser trifft. Land wird durch 1 und Wasser durch 0 dargestellt.

Das Programm muss also die Außenseiten der Quadrate einer Kachel mit den Außenseiten der Quadrate von Nachbarkacheln vergleichen, um festzustellen, ob ein Anlegen möglich ist.

Es darf nur angelegt werden, wenn die Beschaffenheit (1 oder 0) vom Quadrat der Kachel mit den Vergleichsquadrate auf den anderen Kacheln übereinstimmt.

Es sollen unvollständige Kachel-Landschaften vervollständigt werden. Die zu bestimmenden Quadrate sind mit \* gekennzeichnet.

Mit welchen Elementen verglichen wird, hängt ab von wo das Quadrat auf der Kachel liegt. Es gibt 4 Typen.

## Junioraufgabe 2: Parallelen

- oben links → Vergleichselemente: links, oben, links-oben
- oben rechts → Vergleichs-Elemente: rechts, oben, rechts-oben
- unten links → Vergleichs-Elemente: links, unten, links-unten
- unten rechts → Vergleichs-Elemente: rechts, unten, rechts-unten

Nachdem die jeweiligen Vergleichselemente in der Kachel-Landschaft gefunden wurden, muss geschaut werden, ob ein vervollständigen ohne Regelverletzung möglich ist.

Angrenzende Vergleichselemente dürfen entweder unbekannt (\*) sein oder wenn bekannt, müssen die bekannten alle von der selben Art sein (alle bekannten Vergleichselemente sind 1, oder alle bekannten sind 0).

Gibt es sowohl Vergleichselemente die 0 als auch welche, die 1 sind, dann kann nicht vervollständigt werden (Regelverletzung) und es soll eine entsprechende Meldung ausgegeben und das Programm beendet werden.

Sonst soll das Quadrat bestimmt werden als:

- 0 wenn mindestens ein Vergleichselement 0 oder alle \*
- 1 wenn mindestens ein Vergleichselement 1

Zum Schluss soll die vervollständigte Landschaft ausgegeben werden.

## Umsetzung

Das Programm habe ich mit Blockly geschrieben, da ich leider noch keine andere Programmiersprache kann.

Ich habe die zur Verfügung gestellte Eingabe-/Ausgabe Umgebung verwendet. Das Programm sind also die gespeicherten Blöcke als Datei (**Kacheln**), die in die Umgebung geladen werden müssen. Die Eingaben müssen aus den vorgegebenen Textfiles in der kompakten Version (z.b. [map\\_compact.txt](#)) in das Eingabefeld kopiert werden.

Bitte nicht vergessen, die Eingabe mit der „enter“ Taste abzuschließen, da sonst die letzte Zeile nicht eingelesen wird.

Das Programm liest mit der Funktion **einlesen** zuerst die Anzahl der Kachel-Zeilen und Kachel-Spalten ein und speichert sie in den Variablen (**zeilen**, **spalten**). Dann wird die unvollständige Kachel-Landschaft zeilenweise eingelesen und in einer Liste aus Listen (**kachel\_liste**) abgelegt.

Die Funktion **rand** fügt so zusätzliche Elemente in die **kachel\_liste** ein, dass ein Rand um die Landschaft entsteht. Der Rand macht Elemente-Vergleiche bei Randkacheln einfacher.

Im Hauptteil des Programms wird die zu vervollständigende Kachellandschaft Quadrat für Quadrat nach unbekanntem Quadraten (\*) durchsucht. Dies erledigt eine geschachtelte Schleifenstruktur.

Wird ein \* gefunden, wird, mit der Funktion **typ\_bestimmen**, ermittelt, wo sich das Quadrat auf der Kachel befindet (links oben=Typ1, rechts oben = Typ2, links unten = Typ3, rechts unten = Typ4).

Die Funktionen **typ1**, **typ2**, **typ3** und **typ4** finden dann die für den Typ relevanten Vergleichselemente und die Funktion **vergleichen**, wertet die Vergleichselemente aus. Gibt es unter den Vergleichselementen sowohl 1 als auch 0 wird ein Fehlercode („X“) ausgegeben. Sonst, gibt es bei den bekannten Vergleichselementen eine 1, dann wird 1 zurückgegeben, sonst 0.

## Junioraufgabe 2: Parallelen

Wurde beim Vergleichen ein Fehlercode generiert, dann bricht das Programm ab und gibt aus: „Es ist nicht möglich die Kachellandschaft zu vervollständigen!!!“

Sonst wird das neu bestimmte Element in die Kachellandschaft (`kachel_liste`) eingefügt.

Konnte die Landschaft vervollständigt werden, wird sie zum Schluss ausgegeben (`ausgeben`).

Zuvor entfernt die Funktion `randweg` den Hilfsrand wieder.

## Beispiele

### Vorgegebene Beispiele

Das hat mein Programm für die vorgegebenen Beispiele ausgegeben.

Beispiel 1

Eingangsdatei: [map\\_compact.txt](#)

Ausgangslandschaft:

```
10**
01**
**10
**11
```

Vervollständigte Landschaft

```
1000
0110
0110
0111
```

Beispiel 2

Eingangsdatei: [map1\\_compact.txt](#)

Ausgangslandschaft:

```
1001****111001
0110****011001
0110*****1001
1110*****1111
****000001****
****100001****
000110**0111**
011111**1110**
01111111111000
11100000011001
1110*****1001
0111*****0001
```

Vervollständigte Landschaft

```
10011001111001
01100000011001
01100000011001
11100000011111
11100000011111
00011000011110
00011000011110
01111111111000
01111111111000
11100000011001
11100000011001
01111000000001
```

## Junioraufgabe 2: Parallelen

### Beispiel 3

Eingangsdatei: [map2 compact.txt](#)

Ausgangslandschaft:

```
*****
*****
**00**
**10**
**10**
**00**
*****
*****
```

Vervollständigte Landschaft

```
000000
000000
000000
011000
011000
000000
000000
000000
```

### Beispiel 4

Eingangsdatei: [map3 compact.txt](#)

Ausgangslandschaft:

```
10000111100001
11111001111000
1111*****00
0001*****11
0001*****11
1110*****01
11100110000001
1111110000111
```

Vervollständigte Landschaft

```
10000111100001
11111001111000
11111001111000
00011000000111
00011000000111
11100110000001
11100110000001
1111110000111
```

### Beispiel 5

Eingangsdatei: [map4 compact.txt](#)

Ausgangslandschaft:

```
11100000011001111001111000011111
00011001111000011111100000011001
00011001111000011111100000011001
01111111111110000000011001111000
01111111*****000000011001111000
11100000*****011001111001111000
11100000*****011001111001111000
00000000*****111111111110000000
0000000000011111111111110000000
00000000011111111110000001111001
00000000011111111110***01111001
00011001111110000110***11100000
00011001**1110000110***11100000
00011001**000000000***11111111
00011001***00000000011111111111
01111000***0000000111110000000
01111000*****000001111110000000
00000000*****0000000000000000
```

Vervollständigte Landschaft

```
11100000011001111001111000011111
00011001111000011111100000011001
00011001111000011111100000011001
01111111111110000000011001111000
01111111111110000000011001111000
11100000000000011001111001111000
11100000000000011001111001111000
0000000000011111111111110000000
0000000000011111111111110000000
00000000011111111110000001111001
00000000011111111110000001111001
00011001111110000110000111100000
00011001111110000110000111100000
0001100110000000000001111111111
00011001100000000000011111000000
0111100000000000000111110000000
0111100000000000000111110000000
000000000000000000000000000000
```

## Junioraufgabe 2: Parallelen

### Beispiel 6

Eingangsdatei: [map5\\_compact.txt](#)

Ausgangslandschaft:

```
1111111111
1000000111
100000**11
000110**00
**01**1000
**00**0000
**0001**00
**1111**11
0111111111
1110000111
```

Programmausgabe:

```
Es ist nicht möglich
die Kachellandschaft
zu vervollständigen!!!
```

## Eigene Beispiele

### Beispiel 7

Eingangsdatei: eigenes\_beispiel1.txt

Ausgangslandschaft:

```
**
**
```

Programmausgabe:

```
00
00
```

Wenn für ein Element sowohl Land als auch Wasser möglich wäre, wählt mein Programm immer Wasser (0). Das führt hier zu einer reinen Wasser-Landschaft.

### Beispiel 8

Eingangsdatei: eigenes\_beispiel2.txt

Ausgangslandschaft:

```
**11
11**
```

Programmausgabe:

```
0111
1110
```

## Junioraufgabe 2: Parallelen

### Quellcode

Hier die Blöcke von meinem Blockly – Programm. Sie sind in der Datei [Kacheln](#) gespeichert.

```
einlesen
  setze zeilen auf lies Zeile als Zahl
  setze spalten auf lies Zeile als Zahl
  setze kachel_liste auf erzeuge eine leere Liste
  wiederhole bis Ende der Eingabe
  mache
    setze kachel_zeile auf lies Zeile als Text
    setze kachel_zeile auf Liste aus Text erstellen kachel_zeile mit Trennzeichen " "
    in der Liste kachel_liste füge als letztes ein kachel_zeile
```

```
ausgeben
  für jeden Wert p aus der Liste kachel_liste
  mache
    schreibe Text aus Liste erstellen p mit Trennzeichen " "
```

```
rand
  setze randzeile auf erzeuge eine leere Liste
  für jeden Wert q aus der Liste kachel_liste
  mache
    in der Liste q füge als letztes ein " R "
    in der Liste q füge als erstes ein " R "
  wiederhole
    spalten * 2 + 2 mal:
  mache
    in der Liste randzeile füge als letztes ein " R "
  in der Liste kachel_liste füge als letztes ein randzeile
  in der Liste kachel_liste füge als erstes ein randzeile
```

```
randweg
  in der Liste kachel_liste entferne letztes
  in der Liste kachel_liste entferne erstes
  für jeden Wert q aus der Liste kachel_liste
  mache
    in der Liste q entferne letztes
    in der Liste q entferne erstes
```

## Junioraufgabe 2: Parallelen

The image shows two Scratch code snippets. The first, titled 'typ\_bestimmen', uses four 'falls' (if) blocks to determine a value 'typ' based on the parity of variables 'v' and 'w'. The second, titled 'vergleichen mit: x', initializes 'einser' and 'nullen' to 0 and 'element' to an empty string. It then checks for the presence of 'x' at the first, second, and last positions of a list, incrementing 'einser' or 'nullen' accordingly. Finally, it checks if 'einser' is greater than 0 and 'nullen' is greater than 0 to set 'element' to 'X', '1', or '0'.

```
typ_bestimmen
falls (v ist gerade) und (w ist gerade)
  mache setze typ auf 1
falls (v ist gerade) und (w ist ungerade)
  mache setze typ auf 2
falls (v ist ungerade) und (w ist gerade)
  mache setze typ auf 3
falls (v ist ungerade) und (w ist ungerade)
  mache setze typ auf 4
gib zurück typ

vergleichen mit: x
setze einsler auf 0
setze nullen auf 0
setze element auf ""
falls (in der Liste x nimm erstes = "1")
  mache erhöhe einsler um 1
sonst falls (in der Liste x nimm erstes = "0")
  mache erhöhe nullen um 1
falls (in der Liste x nimm #tes 2 = "1")
  mache erhöhe einsler um 1
sonst falls (in der Liste x nimm #tes 2 = "0")
  mache erhöhe nullen um 1
falls (in der Liste x nimm letztes = "1")
  mache erhöhe einsler um 1
sonst falls (in der Liste x nimm letztes = "0")
  mache erhöhe nullen um 1
falls (einsler > 0)
  mache falls (nullen > 0)
    mache setze element auf "X"
  sonst setze element auf "1"
sonst setze element auf "0"
gib zurück element
```

## Junioraufgabe 2: Parallelen

```

typ1
setze lo auf in der Liste in der Liste kachel_liste nimm #tes v - 1 nimm #tes w - 1
setze l auf in der Liste in der Liste kachel_liste nimm #tes v nimm #tes w - 1
setze o auf in der Liste in der Liste kachel_liste nimm #tes v - 1 nimm #tes w
setze Vergleichselemente auf erzeuge Liste mit lo l o
gib zurück Vergleichselemente
  
```

```

typ2
setze ro auf in der Liste in der Liste kachel_liste nimm #tes v - 1 nimm #tes w + 1
setze r auf in der Liste in der Liste kachel_liste nimm #tes v nimm #tes w + 1
setze o auf in der Liste in der Liste kachel_liste nimm #tes v - 1 nimm #tes w
setze Vergleichselemente auf erzeuge Liste mit ro r o
gib zurück Vergleichselemente
  
```

```

typ3
setze lu auf in der Liste in der Liste kachel_liste nimm #tes v + 1 nimm #tes w - 1
setze l auf in der Liste in der Liste kachel_liste nimm #tes v nimm #tes w - 1
setze u auf in der Liste in der Liste kachel_liste nimm #tes v + 1 nimm #tes w
setze Vergleichselemente auf erzeuge Liste mit lu l u
gib zurück Vergleichselemente
  
```

```

typ4
setze ru auf in der Liste in der Liste kachel_liste nimm #tes v + 1 nimm #tes w + 1
setze r auf in der Liste in der Liste kachel_liste nimm #tes v nimm #tes w + 1
setze u auf in der Liste in der Liste kachel_liste nimm #tes v + 1 nimm #tes w
setze Vergleichselemente auf erzeuge Liste mit ru r u
gib zurück Vergleichselemente
  
```

## Junioraufgabe 2: Parallelen

```
einlesen
rand
zähle v von 2 bis Länge von kachel_liste - 1 in Schritten von 1 :
mache
  zähle w von 2 bis 2 * spalten - 1 in Schritten von 1 :
  mache
    setze xy auf in der Liste in der Liste kachel_liste nimm #tes v nimm #tes w
    falls xy = " "
      mache
        setze stelle auf typ_bestimmen
        falls stelle = 1
          mache
            setze bestimmt auf vergleichen mit x typ1
        falls stelle = 2
          mache
            setze bestimmt auf vergleichen mit x typ2
        falls stelle = 3
          mache
            setze bestimmt auf vergleichen mit x typ3
        falls stelle = 4
          mache
            setze bestimmt auf vergleichen mit x typ4
        falls bestimmt = "X"
          mache
            setze v auf Länge von kachel_liste
            setze w auf 2 * spalten + 1
            schreibe " Es ist nicht möglich die Kachellandschaft zu ver! "
        sonst
          in der Liste in der Liste kachel_liste nimm #tes v setze für #tes w ein bestimmt
  randweg
  falls bestimmt ≠ "X"
  mache
    ausgeben
```