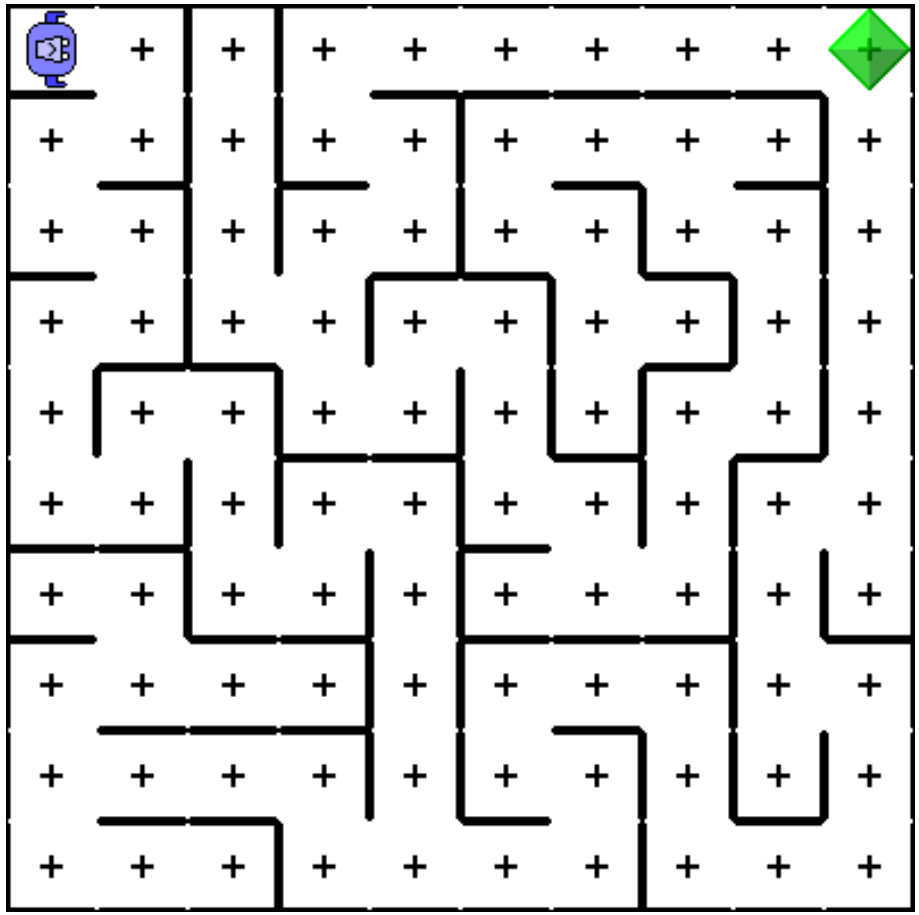
Karel The Robot – Lektion II

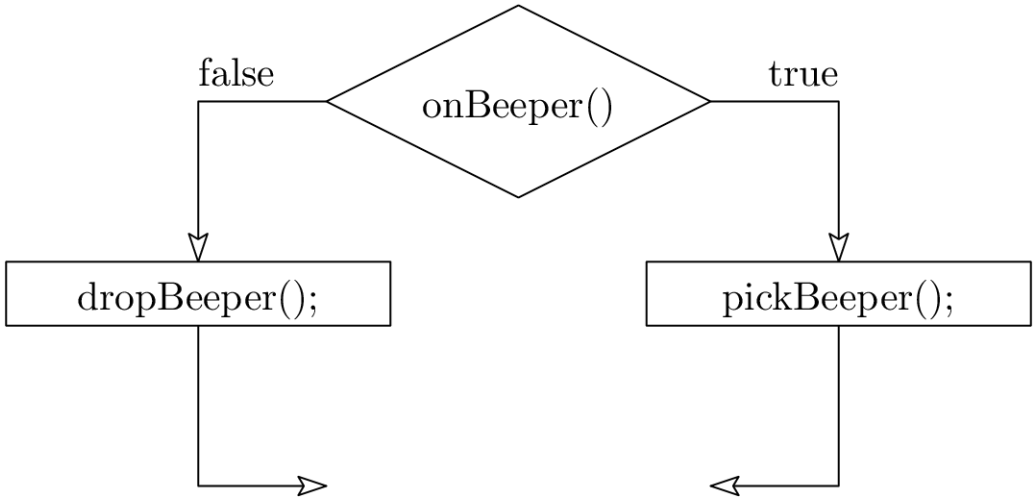
- Bedingte Schleife
- Binärsystem



Bereits bekannt: Die Fallunterscheidung mit Alternative

Wir hatten letzte Woche bereits die Fallunterscheidung kennengelernt:

```
void pickOrDrop()
{
    if (onBeeper())
    {
        pickBeeper();
    }
    else
    {
        dropBeeper();
    }
}
```

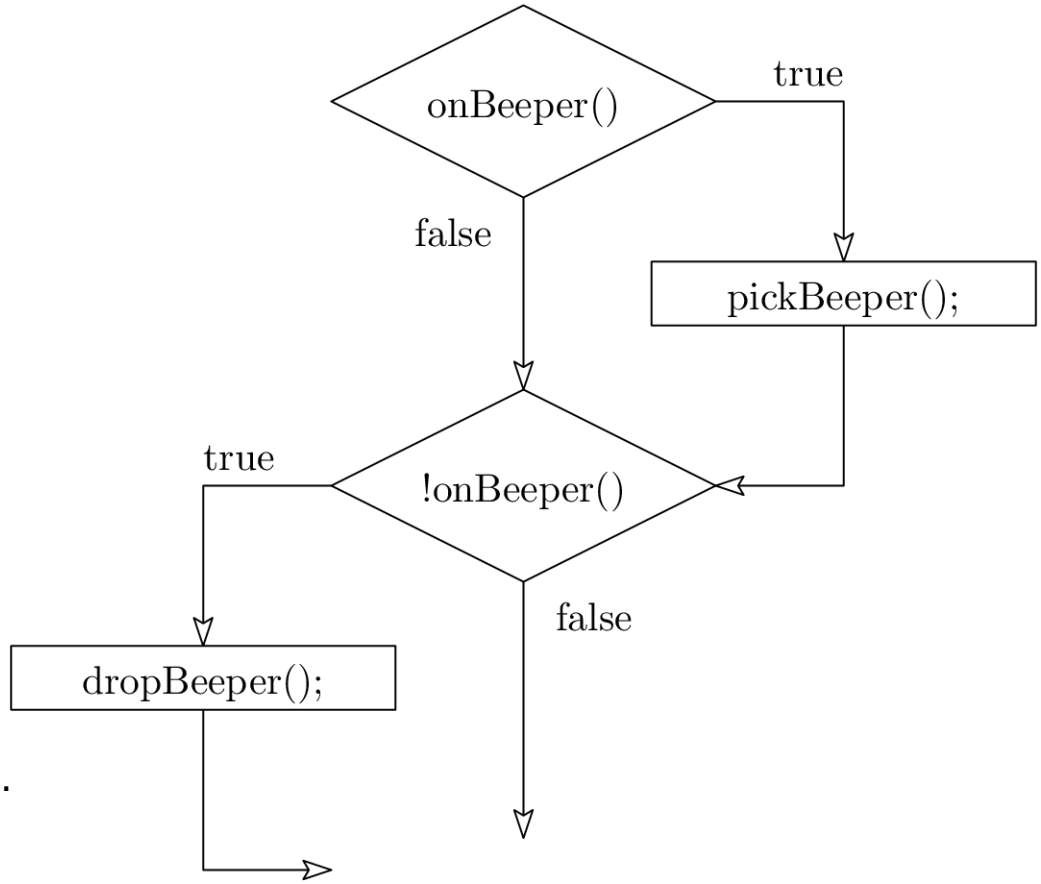


Abhängig von der Bedingung wird einer der beiden Blöcke einmalig ausgeführt.

Eine zweite, negierte Fallunterscheidung statt else

Achtung, das **else** ist im Allgemeinen nicht ersetzbar durch ein negiertes **if**:

```
void pickOrDrop()
{
    if (onBeeper())
    {
        pickBeeper();
    }
    if (!onBeeper())
    {
        dropBeeper();
    }
}
```

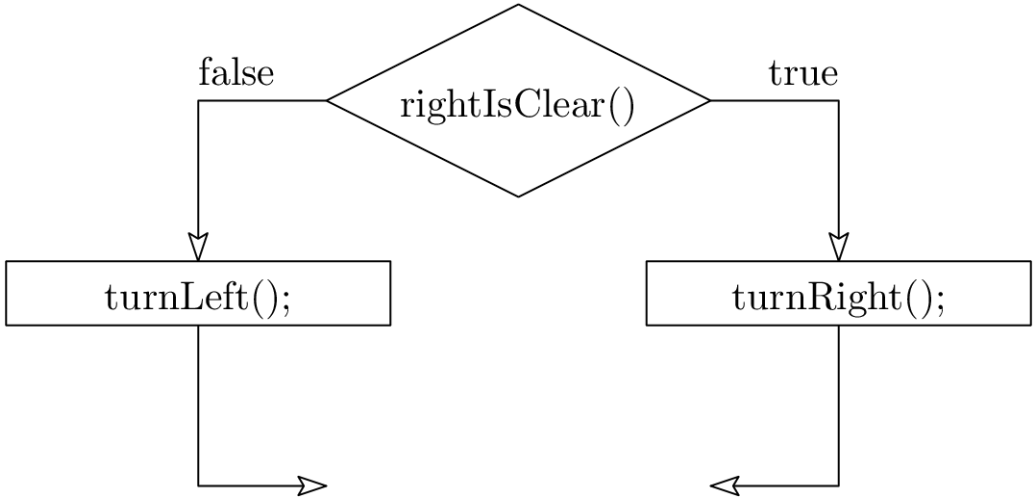


Die zweite Bedingung trifft hier immer zu.
Warum?

Mögliche Pfade durch den Code

Wieviele verschiedene Pfade gibt es durch einen bestimmten Programmabschnitt?

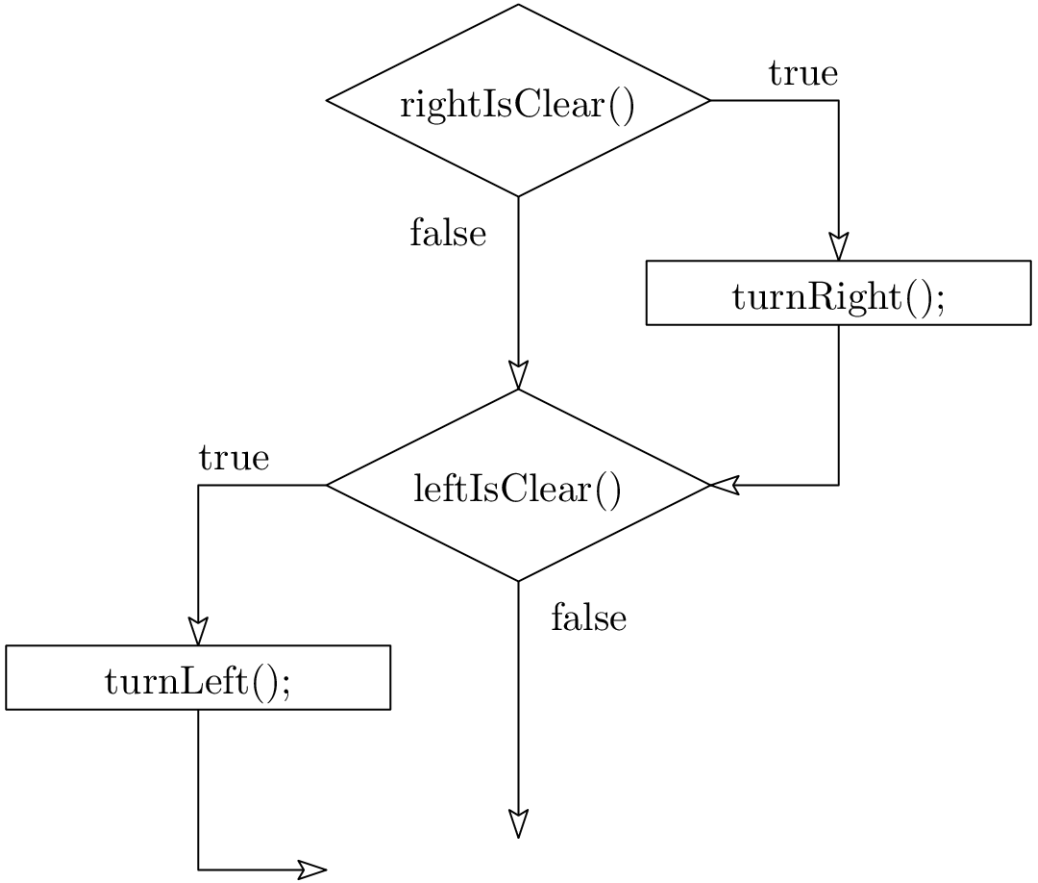
```
void ifelse()
{
    if (rightIsClear())
    {
        turnRight();
    }
    else
    {
        turnLeft();
    }
}
```



Hier gibt es 2 Pfade (entweder wird nur `turnLeft` ausgeführt oder nur `turnRight`).

Mögliche Pfade durch den Code

```
void ifif()
{
    if (leftIsClear())
    {
        turnLeft();
    }
    if (rightIsClear())
    {
        turnRight();
    }
}
```



Hier gibt es 4 Pfade (entweder wird gar nichts ausgeführt oder nur `turnLeft` oder nur `turnRight` oder beides).

Fundstücke aus dem Labor: walkTheLabyrinth

```
void walkTheLabyrinth()
{
    repeat (77)
    {
        if (frontIsClear())
        {
            moveForward();
        }
        if (leftIsClear())
        {
            turnLeft();
            moveForward();
        }
        if (rightIsClear())
        {
            turnRight();
            moveForward();
        }
    }
}
```

In jedem Schleifendurchlauf werden 1-3 `moveForwards` ausgeführt.

Die 77 ist durch Ausprobieren rausgefunden worden. Geht das nicht schöner?

Musterlösung: walkTheLabyrinth

Das Labyrinth besteht aus 100 Feldern, und Karel steht bereits auf dem ersten Feld. Also muss Karel 99 Schritte machen. Es erscheint daher logisch, eine Schleife mit 99 Wiederholungen zu verwenden, und jedes Mal genau einen Schritt zu gehen:

```
void walkTheLabyrinth()
{
    repeat (99)
    {
        turnAwayFromWall();
        moveForward();
    }
}
```

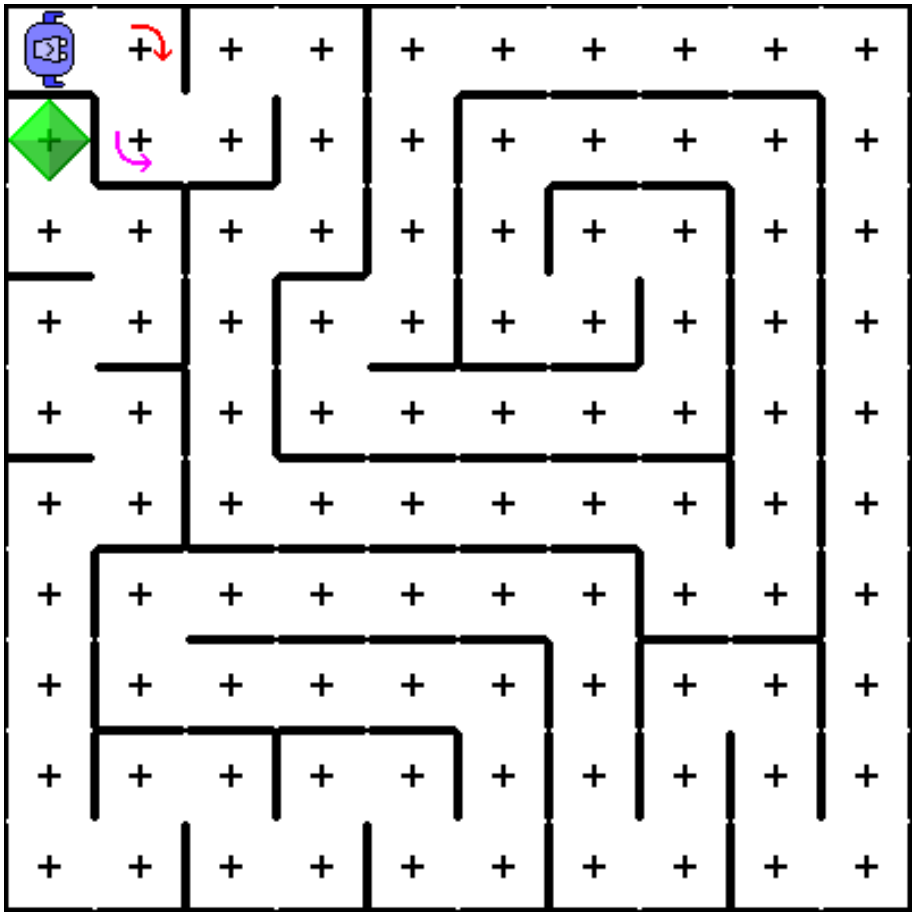
So sieht man auf den ersten Blick, dass Karel exakt 99 Schritte macht.

Jetzt müssen wir uns nur noch darum kümmern, dass Karel sich korrekt dreht.

Musterlösung: walkTheLabyrinth

Karel muss sich entweder gar nicht drehen, nach links oder nach rechts:

```
void turnAwayFromWall()
{
    if (frontIsClear())
    {
    }
    else
    {
        if (leftIsClear())
        {
            turnLeft();
        }
        else
        {
            turnRight();
        }
    }
}
```



Wieviele Pfade gibt es durch diesen Befehl?

Musterlösung: walkTheLabyrinth

Den leeren then-Block kann man eliminieren, indem man die Bedingung negiert:

```
void turnAwayFromWall()
{
    if (!frontIsClear())
    {
        if (leftIsClear())
        {
            turnLeft();
        }
        else
        {
            turnRight();
        }
    }
}
```

Es gibt nach wie vor 3 Pfade durch den Befehl.

Fundstücke aus dem Labor: removeTheTiles

```
void removeTheTiles()
{
    repeat (118)
    {
        if (onBeeper())
        {
            pickBeeper();
        }
        if (beeperAhead())
        {
            moveForward();
        }
        else
        {
            turnLeft();
        }
    }
    moveForward();
}
```

Wo kommt die 118 her? Wieviele Pfade gibt es durch den Schleifenrumpf?

Musterlösung: removeTheTiles

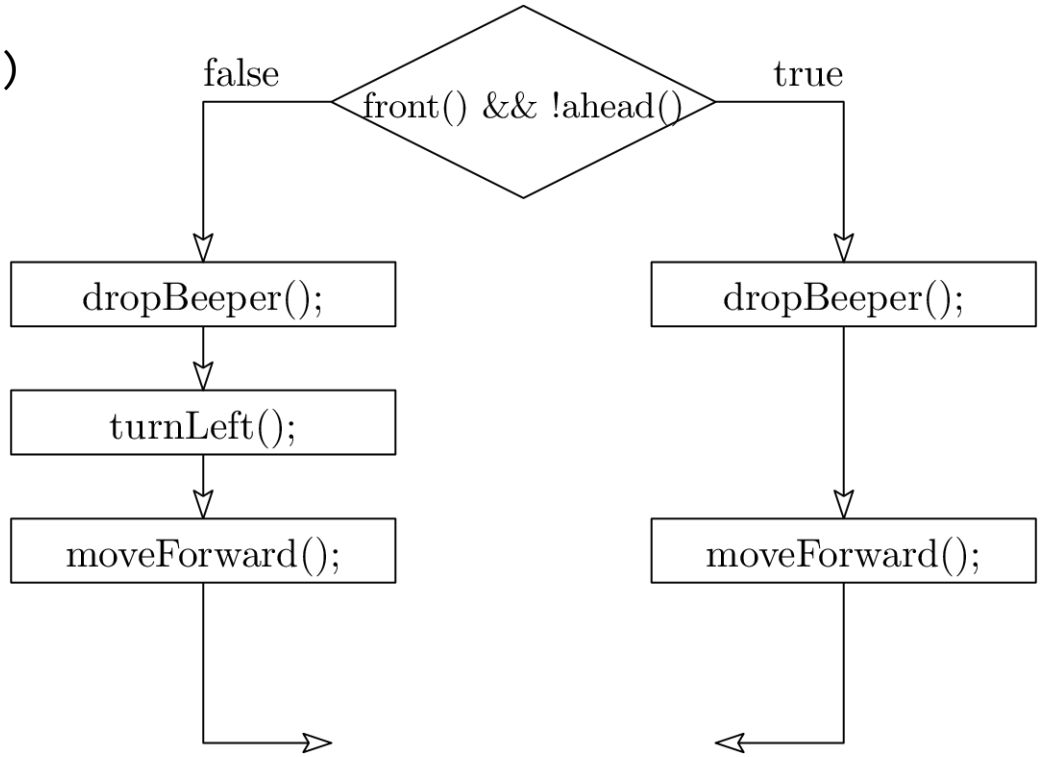
Karel soll 100 Diamanten aufsammeln. Es erscheint daher logisch, eine Schleife mit 100 Wiederholungen zu verwenden, und jedes Mal einen Diamanten aufzusammeln:

```
void removeTheTiles()
{
    repeat (100)
    {
        pickBeeper();
        faceTile();
        moveForward();
    }
}

void faceTile()
{
    if (!beeperAhead())
    {
        turnLeft();
    }
}
```

Fundstücke aus dem Labor: tileTheFloor

```
if (frontIsClear() && !beeperAhead())
{
    dropBeeper();
    moveForward();
}
else
{
    dropBeeper();
    turnLeft();
    moveForward();
}
```

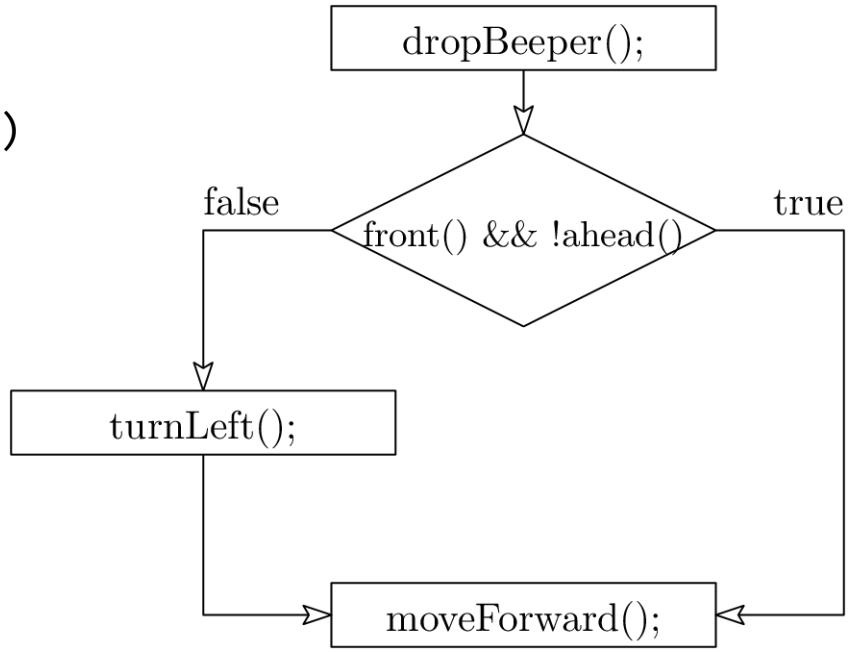


Beobachtung: Die beiden Blöcke sind sich recht ähnlich.

Frage: Kann man die Fallunterscheidung vereinfachen?

Fundstücke aus dem Labor: tileTheFloor

```
dropBeeper();
if (frontIsClear() && !beeperAhead())
{
}
else
{
    turnLeft();
}
moveForward();
```



Beobachtung: Eine Fallunterscheidung mit einem leeren Block ist eher ungewöhnlich.

Frage: Kann man das **else** loswerden, d.h. das **if/else** durch ein einfaches **if** ersetzen?

Fundstücke aus dem Labor: tileTheFloor

```
// vorher:
if (frontIsClear() && !beeperAhead())
{
}
else
{
    turnLeft();
}

// nachher:
if (!(frontIsClear() && !beeperAhead()))
{
    turnLeft();
}
```

Jetzt haben wir zwei Negationen; geht das nicht einfacher?

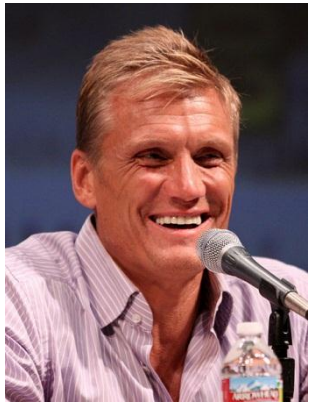
Intermezzo: Blondinen

- Eine Blondine ist weiblich **und** blond.

- Was gilt für alle Menschen,
  die *nicht* Blondinen sind? Sie sind:

  a) nicht weiblich **und** nicht blond

  b) nicht weiblich **oder** nicht blond
  (logisches *oder* erlaubt auch beides)



Die Gesetze von Augustus De Morgan

```
!(a && b) = !a || !b
```

Treffen beide Bedingungen zu? Nein?
Dann trifft mindestens eine Bedingung nicht zu!

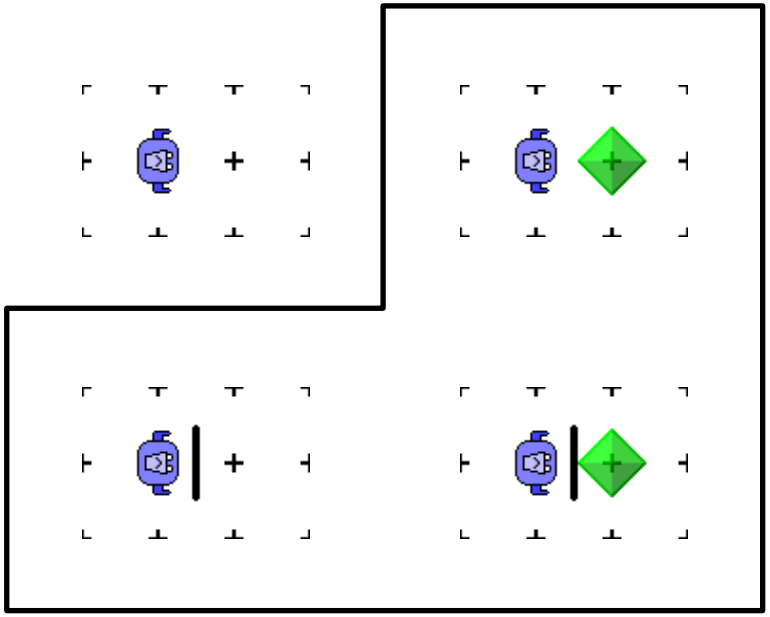
```
!(a || b) = !a && !b
```

Trifft mindestens eine Bedingung zu? Nein?
Dann treffen beide Bedingungen nicht zu!

Auf unser Beispiel angewendet bedeutet das:

```
if (!(frontIsClear() && !beeperAhead()))
```

```
if ( !frontIsClear() ||  beeperAhead())
```



(Der Fall rechts unten kommt in `tileTheFloor` nicht vor.)

Bereits bekannt: Die Zählschleife

Wir hatten bereits gesehen, dass man einen Block mehrfach ausführen kann:

```
void moveAcrossWorld()
{
    repeat (9)
    {
        moveForward();
    }
}
```

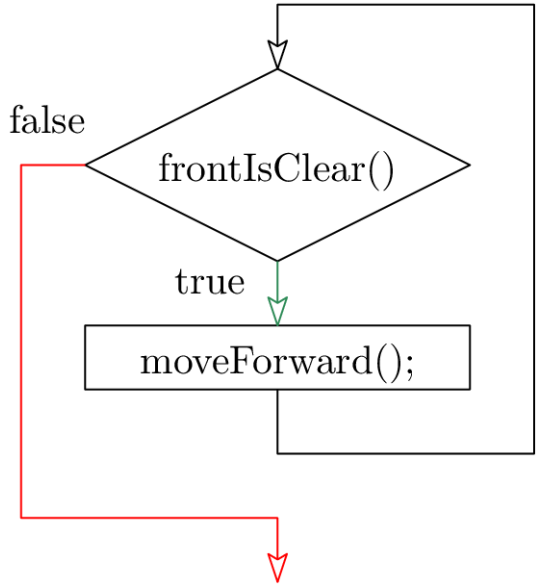
Das funktioniert natürlich nur, falls

1. die Welt genau 10 Felder breit ist und
2. Karel am anderen Ende der Welt steht.

Neu: Die bedingte Schleife

Falls man beim Schreiben des Programms noch nicht weiß, wie oft der Rumpf später einmal durchlaufen werden soll, bietet sich stattdessen die bedingte Schleife an:

```
void moveToWall()
{
    while (frontIsClear())
    {
        moveForward();
    }
}
```



Solange die Bedingung wahr ist, wird der Block immer wieder komplett ausgeführt. Falls die Bedingung bereits am Anfang falsch ist, wird der Block überhaupt nicht ausgeführt.

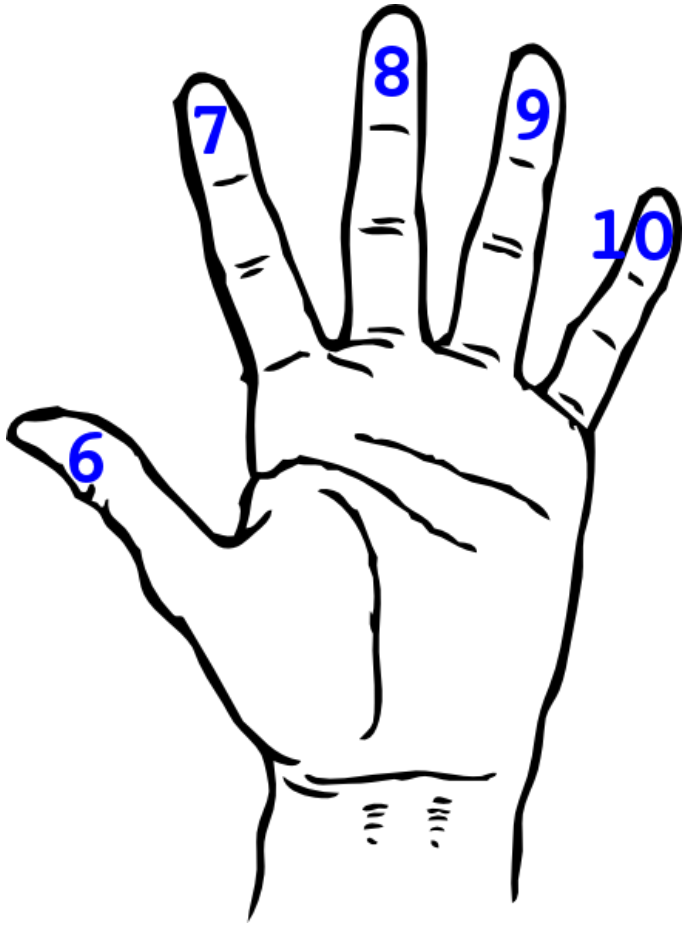
Die Schleifenbedingung wird immer nur vor dem nächsten Durchlauf des gesamten Blocks geprüft, und nicht etwa nach jedem einzelnen Schritt innerhalb des Blocks!

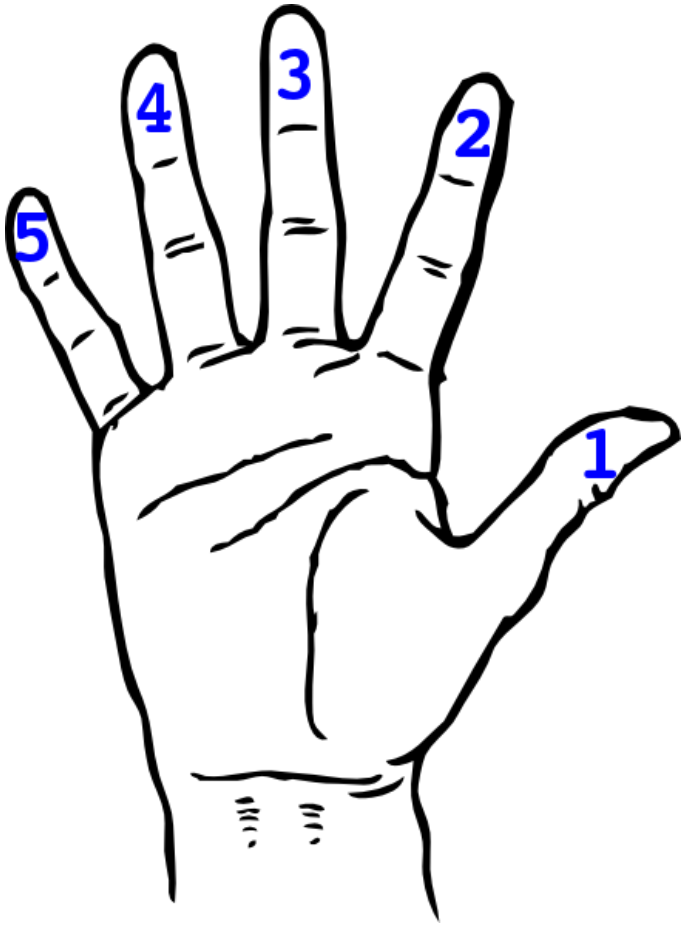
Ein vertrautes Stellensystem: Das Dezimalsystem

Zehntausender	Tausender	Hunderter	Zehner	Einer
2	0	5	9	7
2	0	5	9	8
2	0	5	9	9
2	0	6	0	0
2	0	6	0	1

$$
\begin{array}{cccccc}
 & 3 & 1 & 4 & 1 & 5 \\
+ & 7 & 1 & 8 & 2 & 8 \\
\hline
1 & 0 & 3 & 2 & 4 & 3
\end{array}
\qquad
\begin{array}{cccccc}
1 & & 1 & & 1 & \\
\end{array}
$$

Warum gibt es eigentlich zehn verschiedene Ziffern?





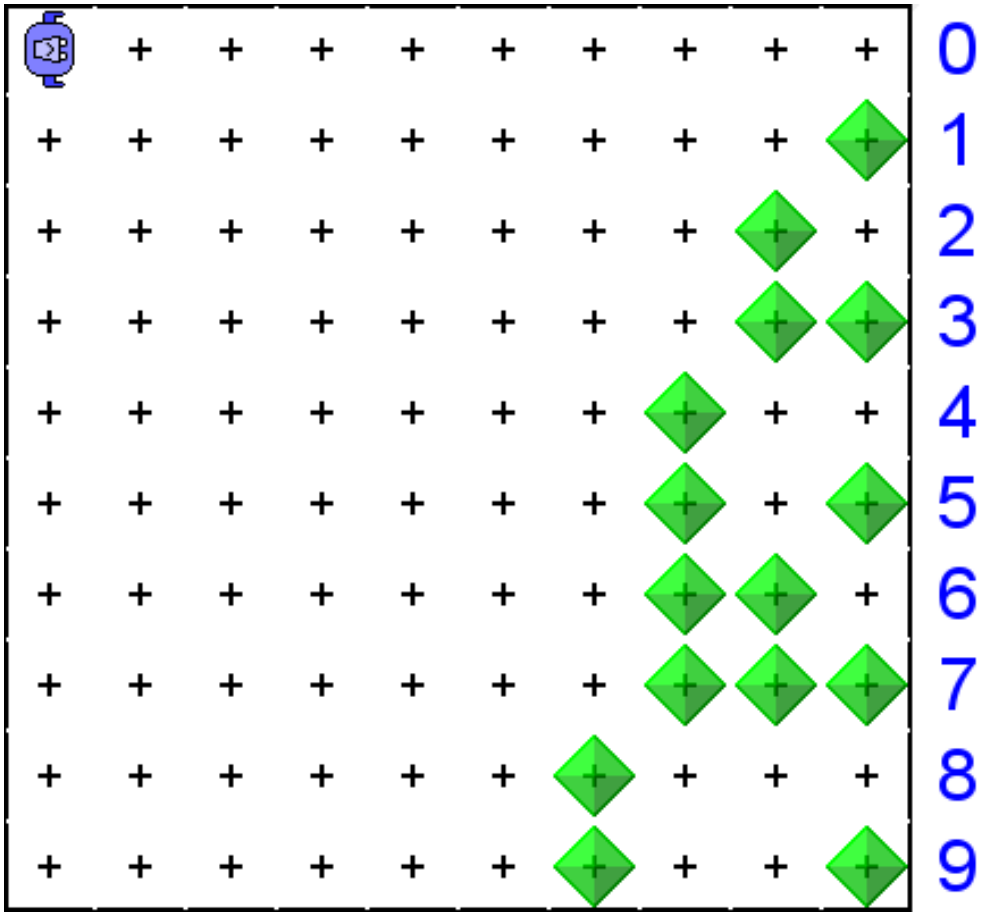
Ein minimalistisches Stellensystem: Das Binärsystem

Sechzehner	Achter	Vierer	Zweier	Einer
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0

$$
\begin{array}{cccccc}
 & 0 & 1 & 1 & 0 & 0 \\
+ & 0 & 1 & 1 & 1 & 0 \\
 & {\scriptstyle 1} & & {\scriptstyle 1} & & \\
\hline
 & 1 & 1 & 0 & 1 & 0
\end{array}
$$

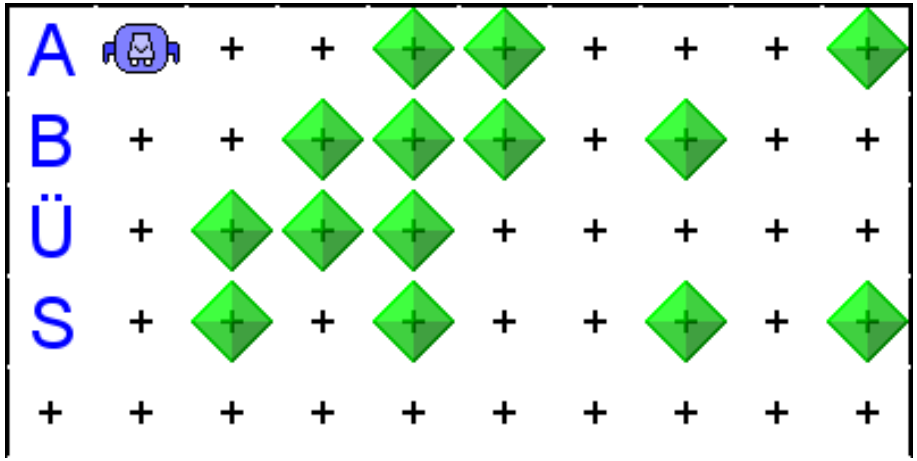
Das Binärsystem, realisiert in Karels Welt

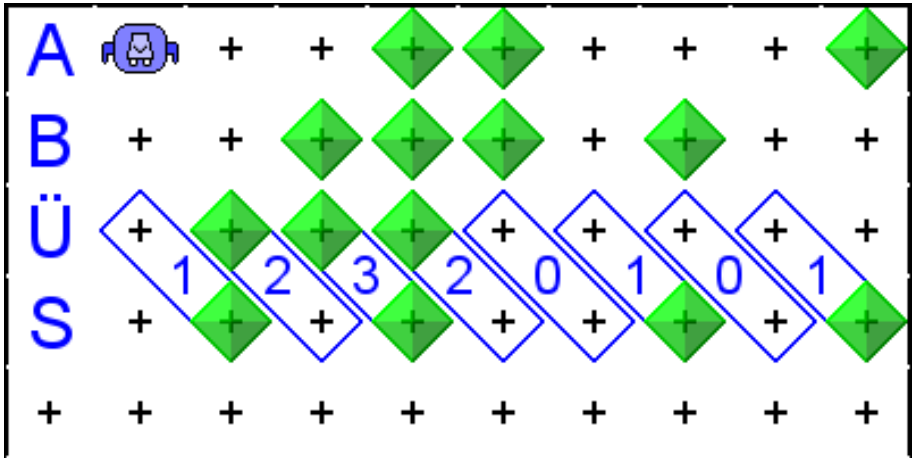
- Karel ist trotz seiner eingeschränkten Möglichkeiten mächtig genug, alle berechenbaren Funktionen (z.B. die Addition) zu berechnen.
- Frage: Wie kann das funktionieren, wenn Karel gar keine Zahlen kennt?
- Ansatz: Wir kodieren Binärzahlen durch das Ablegen von Diamanten!



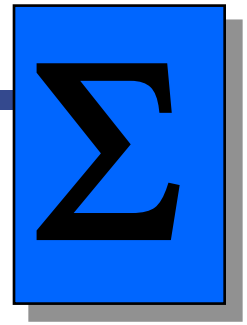
Schriftliche Addition im Binärsystem

- Addition funktioniert im Binärsystem genau so wie im Dezimalsystem.
- Die Ziffern (A, B) werden spaltenweise von rechts nach links aufsummiert (S), wobei man evtl. auftretende Überträge (Ü) bei der nächsten Stelle berücksichtigt.
- Wenn man seinen Kopf um 45° nach rechts neigt, dann sieht man im blauen Kasten das binäre Ergebnis der Addition A+B+Ü einer Spalte (00 = 0, 01 = 1, 10 = 2, 11 = 3), also die Anzahl der Diamanten in den ersten drei Zeilen einer Spalte.





Zusammenfassung



- Fallunterscheidungen, die untereinander stehen, werden unabhängig voneinander ausgeführt. Oft ergeben sich dadurch viele vom Programmierer unbeabsichtigte Pfade durch den Code.
- Die DeMorgan‘schen Regeln helfen uns bei der Vereinfachung von negierten zusammengesetzen Bedingungen.
- Die bedingte Schleife erlaubt es Karel, Befehle zu wiederholen, solange eine bestimmte Bedingung erfüllt ist.
- Karel kann mit Zahlen im Binärsystem rechnen, indem er diese durch Diamanten kodiert, die er in der Welt platziert.
- Die schriftliche Addition funktioniert im Binärsystem (und in allen anderen Stellensystemen) genau so wie im Dezimalsystem.