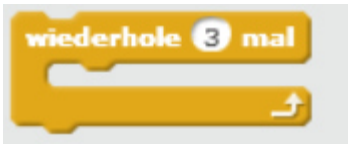
6 Wiederholbefehle

In nahezu jedem Programm gibt es Sequenzen, die es notwendig machen mehrere Befehle, z.T. auch gleiche Befehle, hintereinander auszuführen. Bisher haben wir die notwendige Anzahl Befehle geschrieben. Die nachfolgenden Befehle LOOP und WHILE erweitern noch einmal den Wortschatz des Roboters für wiederholte Ausführung von Befehlen oder Befehlsblöcken.

6.1 LOOP (REPEAT, FOR)

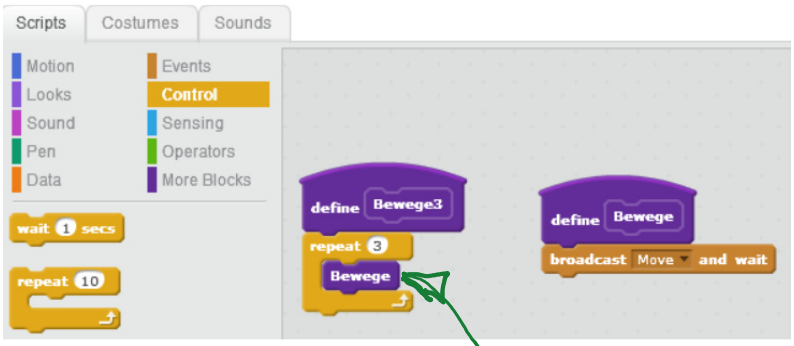
Den LOOP-Befehl (repeat, dt.: wiederhole eine **fixe** Anzahl von Durchläufen) haben wir schon weiter oben behandelt und verwendet; hier noch einmal eine Wiederholung.
Anstelle der drei Aufrufe von **Bewege** kann man auch eine Schleife (loop) verwenden. In Scratch wird eine Schleife mit einer fixen Anzahl von Durchläufen in Steuerung engl. Control durch den Block:



oder engl.


verwendet, wobei die Anzahl der Durchläufe (hier: 3) im weissen Kreis eingetragen wird.

Damit ergibt sich für die obere Funktion: **Bewege3** folgender Code mit gleicher Aufgabe die Funktion Move über Bewege drei Mal aufzurufen:



In vielen Programmiersprachen, wie z.B. auch in C/C++, wird FOR dafür verwendet, wenn die Anzahl der Durchläufe bei Schleifenbeginn bekannt ist.
Die FOR-Schleife gibt die Möglichkeit, dass der Roboter einen Befehl eine bestimmte Anzahl, die im Vorhinein bekannt ist, so oft ausführt (Syntax):

```
for (Laufvariable initialisieren; Abbruchbedingung; Laufvariablen erhöhen) {
      Anweisungen im SCHLEIFENKÖRPER;
}
```

Z.B. wie oft wird die Funktion Bewege aufgerufen?

```
for( int i=0; i < 3; i++ )
      Bewege();
```

Nested-For (ineinander geschachtelte Schleifen): Wie oft wird nun die Funktion Bewege aufgerufen? Beachte, dass die **geschwungenen Klammern {}** weggelassen werden können, wenn nur ein Befehl im Schleifenkörper ist.

```
for( int i=0; i < 3; i++ )
      for( int j=0; j < 3; j++ )
            Bewege();
```

```
for( int i=0; i < 3; i++ ) {
      for( int j=0; j < 3; j++ ) {
            Bewege();
      }
}
```

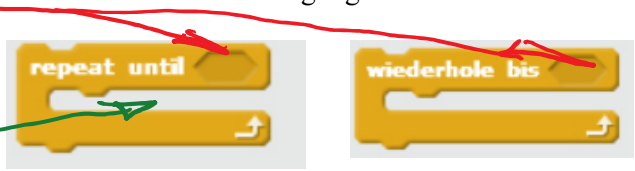
Siehe Blockly < https://developers.google.com/blockly/ > für JavaScript, Phyton und Dart (C-ähnlich).

6.2 WHILE-Schleife

Bei der WHILE-Schleife ist die Anzahl der Durchläufe bei Schleifenbeginn noch nicht bekannt. Allerdings ist daher die Abbruchbedingung sehr wichtig und sollte sehr genau auf die Korrektheit überprüft werden. Für die Abbruchbedingung stehen wieder die Befehle von der Bertl Bibliothek zur Verfügung:

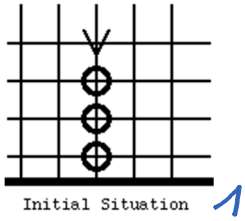
Syntax:

```
while ( Abbruchbedingung ) {
      Anweisungen im SCHLEIFENKÖRPER;
}
```

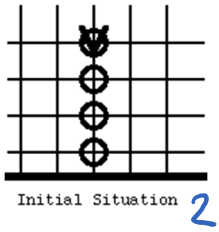


Die geschweiften Klammern {} können wieder entfallen, wenn nur eine Anweisung im Schleifenkörper ausgeführt werden soll.

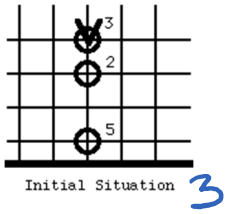
Aufgabe PickAll-Funktion
Aus der Ausgangssituation (Initial Situation 1) in der Abbildung schreiben Sie eine Funktion PickAll(), die alle Beeper in einer Schleife bis zur Wand aufheben soll.
Welche Fragen/Probleme treten auf.



Funktioniert die Funktion PickAll() auch noch mit der nächsten Initial Situation 2, wo Bertl schon auf einem Beeper steht? Was ist im Schleifenkörper zu ändern? Funktioniert es nun?



Was ist für die 3. Initial Situation zu ändern?

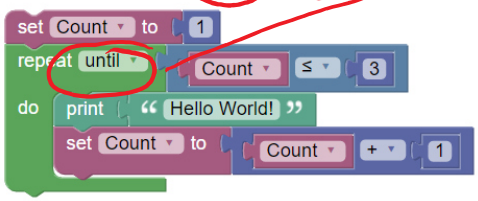


Allerdings gibt es in den meisten gebräuchlichen Programmiersprachen den Befehl REPEAT UNTIL (also WIEDERHOLE solange BIS die Bedingung eintrifft nicht.

Sondern nur WHILE (bzw. DO{ .. }WHILE), sodass die Abbruchbedingung INVERTIERT werden muss. (Siehe BLOCKLY von Google < https://developers.google.com/blockly/ >


```
Count = 1;
while (Count <= 3) {
        print('Hello World!');
        Count = Count + 1;
}
```

Ändert man das while auf until ergibt sich folgender Code in der Abbruchbedingung (Beachte die Negation !)




```
Count = 1;
while (!(Count <= 3)) {
        print('Hello World!');
        Count = Count + 1;
}
```

In Phyton steht für das Rufzeichen ! z.B. das not

```
Count = 1
while not Count <= 3:
  print('Hello World!')
  Count = Count + 1
```

6.3 Die Abbruchbedingung in Programmiersprachen C/C++, JavaScript, …

Bei der Schleife wird eine Abbruchbedingung geprüft, mit der entschieden wird, ob der Schleifenkörper (Schleifeninhalt) ausgeführt wird. Da im Gegensatz zu den Auswahlbefehlen eine fehlerhafte Abbruchbedingung zu einer Endlosschleife führen könnte, ist auf die Erzeugung gültiger Abbruchbedingungen besonderes Augenmerk zu legen.

WICHTIG:
Die nachfolgenden 5 Schritte sollen dazu beitragen eine korrekte Abbruchbedingung zu erzeugen und sind Ihnen wärmstens ans Herz gelegt:

1. **#Zielbedingung formulieren**
2. **#Zielbedingung invertieren (ev. De Morgan)**
3. **#Im Schleifenkörper sich der Beendigung der Bedingung nähern**
4. **#Vor der Schleife: Variablen der Abbruchbedingung Initialisieren**
5. **#Nach der Schleife: ev. Variablen zurücksetzen**

Z.B. soll der Roboter sich solange weiterbewegen bis er auf einen Beeper trifft.

1. **#Zielbedingung formulieren:** Der Roboter soll wann stehen bleiben? Wenn er auf einem Beeper steht: NextToABeeper() == 1

2. **#Zielbedingung invertieren:** solange er NICHT (NOT) auf einem Beeper steht soll der Schleifenkörper ausgeführt werden: !(NextToABeeper() == 0)

3. **# Im Schleifenkörper sich der Beendigung der Bedingung nähern:** Im Schleifenkörper muss es Befehle geben, sodass diese irgendwann die Bedingung der Schleife abbricht: Move();

4. **#Vor der Schleife: Variablen der Abbruchbedingung Initialisieren:** Hier muss der Roboter korrekt in die Welt gesetzt werden.
5. **#Nach der Schleife: eventuell Variablen zurücksetzen:** Hier nichts zu tun, kann entfallen.

```
GoToBeeper() {
      while( !NextToABeeper() ) {
            Move();
      }
}
```

In C/C++: Die Integervariable var soll bis 10 von 0 hoch zählen

1. **Zielbedingung:** var == 10 → fertig
2. **Zielbedingung invertieren: !(var == 10) → entspricht (var != 10)→ solange (while) var nicht gleich 10**

```
int var = 0;      // 4. Vor der Schleife: Variable var auf 0 initialisieren

while( var != 10 ) {
      var++;      // 3. Der Zielbedingung var = 10 nähern (hochzählen: var = var + 1)
}
var = 0;          // 5. Integervariable var zurücksetzen auf 0 (falls notwendig)
```

Folgende Bedingungen funktionieren ebenso:

1. **Zielbedingung:** var >= 10 → fertig
2. **Zielbedingung invertieren: !(var >= 10) → entspricht (var < 10) → solange (while) var kleiner 10**

De Morgan: Aus „<=“ wird „>“; >= → <; < → >=; > → <=; == → !=; != → == (Vergleichsoperatoren)
Aus „&&“ wird „||“; || → && (Logische Operatoren)