3. Algorithmen

In diesem Kapitel werden Kontrollstrukturen erarbeitet und ihr Einsatz eingeübt. Da die Teilkapitel aufeinander aufbauen, werden zur Bearbeitung der Aufgaben z. T. auch Kenntnisse aus den jeweiligen vorangegangenen Teilkapiteln benötigt. Die Komplexität der Aufgaben steigert sich also im Verlauf des Kapitels.

3.1 Kontrollstrukturen/3.2 Wiederholungen

SB S. 41

1. a) mögliche Bedingungen sind z. B. Lichtintensität größer als ..., Bodenbeschaffenheit (Steigung kleiner als ..., Festigkeit), Hindernis vorn, Temperatur, Akkustatus, Signale von der Bodenstation etc.

b) z. B.: wenn kein Hindernis vorne ist und der Akku mehr als halb voll ist, fahre vorwärts. Wenn ein Hindernis vorne ist, umfahre es. Wenn der Akku halb leer ist, kehre zum Ausgangspunkt zurück.

2. a) Ist ein Hügel rechts vom Rover? → huegelVorhanden("rechts")
Ist ein Hügel vor dem Rover? → huegelVorhanden("vorne")
Liegt ein Gestein auf dem aktuellen Feld? → gesteinVorhanden()
Ist der Ausgangspunkt wieder erreicht? → markeVorhanden()

b) 1. Lege eine Marke ab.
2. fahre ein Feld vorwärts.
3. Führe die folgenden Schritte aus, bis eine Marke am Platz ist.
    4. Solange rechts ein Hügel und vorne kein Hügel ist, fahre vorwärts,
        5. sammle dabei alle Gesteine auf.
    6. Wenn rechts kein Hügel mehr ist, drehe nach rechts.
    7. Wenn vorne ein Hügel ist, drehe nach links.
    8. Beginne wieder bei 3.

SB S. 42

3.
```
public void bergSeiteAbfahren()
{
    while(huegelVorhanden("rechts"))
    {
        fahre();
    }
}
```

4. a) Der Rover fährt fünf Felder vorwärts. Die Bedingung wird sechsmal überprüft.

b) Der Rover bleibt stehen. Die Bedingung wird einmal überprüft.

c) Der Rover fährt ein Feld vorwärts. Die Bedingung wird zweimal überprüft.

5. a)
```
while(gesteinVorhanden())
{
  analysiereGestein();
  fahre();
}
```

b)
```
while(gesteinVorhanden())
{
  analysiereGestein();
  setzeMarke();
  fahre();
}
```

SB S. 43

6. Nachprüfende Wiederholung

a) Individuelle Lösungen

b)

vorprüfend	nachprüfend

`while(huegelVorhanden("vorne"))`
`{`
`  drehe("rechts");`
`}`
	`do`
	`{`
	`  fahre();`
	`}while(huegelVorhanden("vorne"))`

`while(!gesteinVorhanden())`
`{`
`  fahre();`
`}`
	`do`
	`{`
	`  fahre();`
	`}while(!gesteinVorhanden())`

`while(!huegelVorhanden("rechts"))`
`{`
`  fahre();`
`}`
	`do`
	`{`
	`  fahre();`
	`}while(!huegelVorhanden("rechts"))`

7. a) Der korrekte Quelltext muss lauten:

```
while(!huegelVorhanden("vorne"))        // Anführungszeichen fehlten
                                        // schließende Klammer fehlte
{                                       // geschweifte Klammer fehlte
  fahre();
}
drehe("rechts");
while(!gesteinVorhanden())              // Methodenklammern fehlten
{
fahre();                                // Semikolon fehlte
}
analysiereGestein();
```

b) Individuelle Lösungen

SB S. 44

8. a)

```
public void bergketteAbfahren()
{
  while(huegelVorhanden("vorne"))
  {
      drehe("links");
      fahre();
      drehe("rechts");
      fahre();
  }
}
```

3.3 Zählschleifen

SB S. 46

1. a)

Zähler	Schleifenbedingung erfüllt?	Ausgeführte Anweisung	Zähler aktualisieren
i=0	wahr, da 0<4	fahre(); analysiereGestein();	i=0+1=1
i=1	wahr, da 1<4	fahre(); analysiereGestein();	i=1+1=2
i=2	wahr, da 2<4	fahre(); analysiereGestein();	i=2+1=3
i=3	wahr, da 3<4	fahre(); analysiereGestein();	i=3+1=4
i=4	falsch, da 4 nicht <4	–	–

b) z. B.

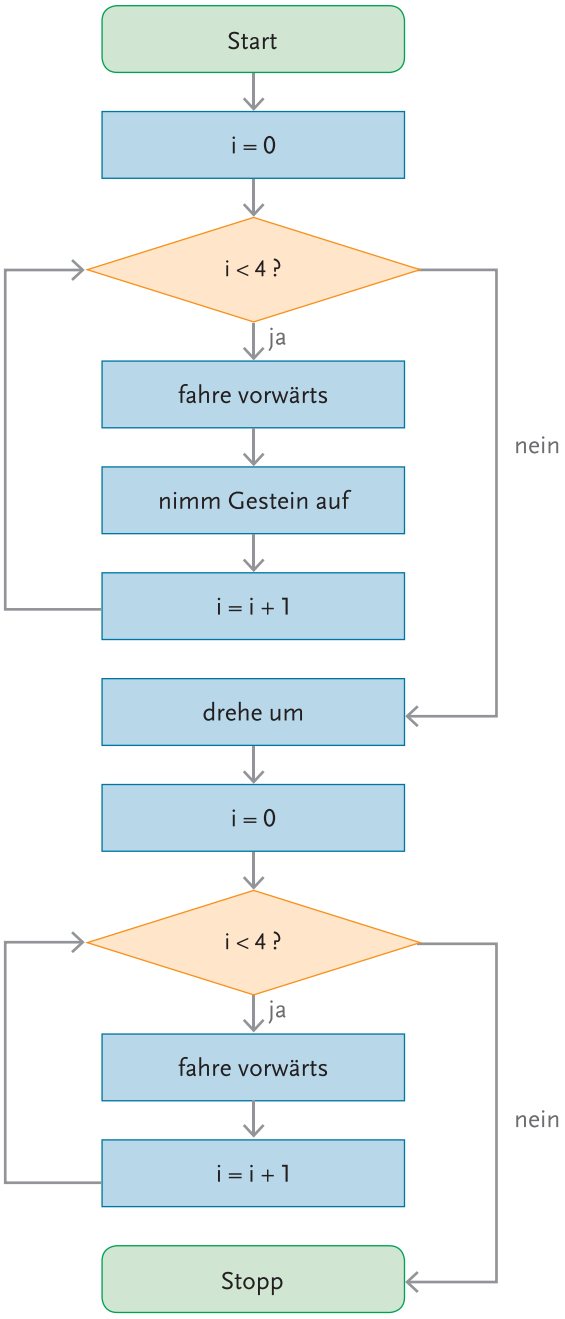
```
for(int i=0; i<4; i--)
{
  fahre();
  analysiereGestein();
}
```

2. a)

```
public void nimmVierGesteinsprobenAuf()
{
  for(int i=0; i<4; i++)
  {
      fahre();
      analysiereGestein();
  }
}
```

b)

```
public void nimmVierGesteinsprobenAuf()
{
  for(int i=0; i<4; i++)
  {
      fahre();
      analysiereGestein();
  }
  drehe("rechts");
  drehe("rechts");
  for(int i=0; i<4; i++)
  {
      fahre();
  }
}
```



3. a)

```
public void methode3_3_3a()
{
  while(!gesteinVorhanden())
  {
      fahre();
  }
  for(int i=0; i<5; i++)
  {
      analysiereGestein();
      fahre();
  }
}
```

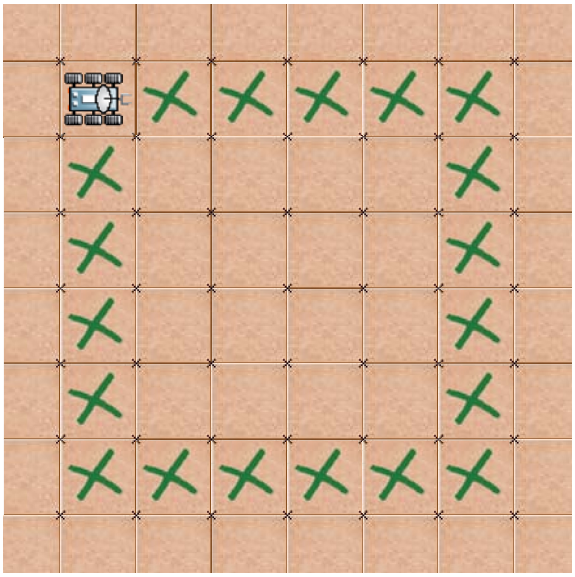
b) Der Schleifenkopf muss geändert werden zu: `for(int i=23; i<28; i++)`

c) Der Schleifenkopf muss geändert werden zu: `for(int i=5; i>0; i--)`

d)
```
public void methode3_3_3d()
{
  setzeMarke();
  while(!gesteinVorhanden())
  {
      fahre();
  }
  for(int i=0; i<5; i++)
  {
      analysiereGestein();
      fahre();
  }
  drehe("rechts");
  drehe("rechts");
  while(!markeVorhanden())
  {
      fahre();
  }
}
```

SB S. 47

4. a)



b)
```
public void methode3_3_4b()
{
  for(int i = 0; i < 2; i++)
  {
      for (int j = 0; j < 4; j++)
      {
          setzeMarke();
          fahre();
      }
      drehe("rechts");
      for (int k = 0; k < 2; k++)
      {
          setzeMarke();
          fahre();
      }
      drehe("rechts");
  }
}
```

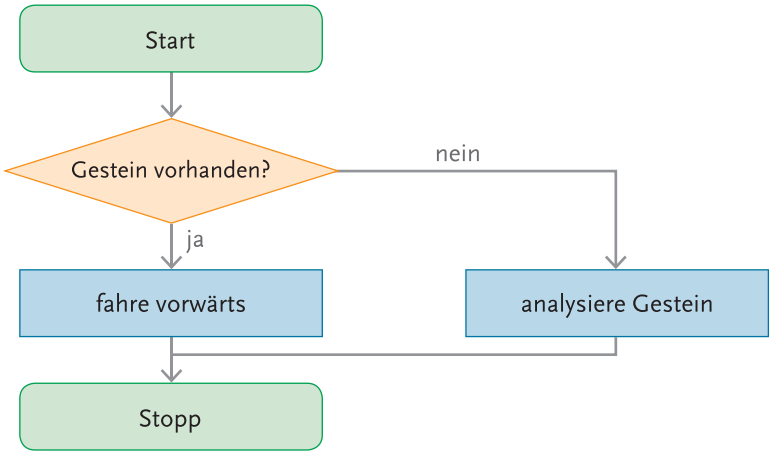
c)
```
//eckige Spirale mit höchster Seitenlänge grenze (z.B. 8)
public void methode3_3_4c(int grenze)
{
  for (int s = 1; s < grenze; s++)  // s = aktuelle Seitenlänge
  {
      for (int i = 0; i < s; i++)   // lege eine Seite
      {
          setzeMarke();
          fahre();
      }
      drehe("rechts");
  }
}
```

d) Individuelle Lösungen

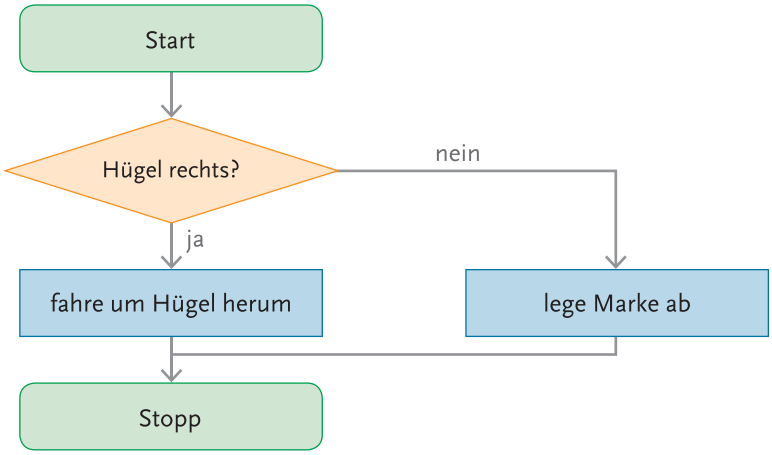
3.4 Bedingte Anweisungen

SB S. 49

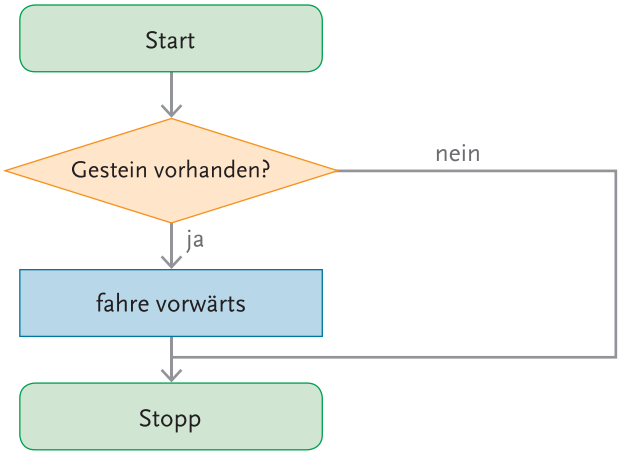
1. a) Falls sich auf dem Feld des Rovers kein Gestein befindet, so soll er ein Feld vorwärtsfahren. Falls dort jedoch ein Gestein ist, so soll er es analysieren (Äquivalent zu Beispiel im Lehrtext).



b) Falls sich rechts vom Rover ein Hügel befindet, so soll er einmal um ihn herumfahren. Ansonsten soll er eine Marke ablegen.



c) Falls sich auf dem Feld des Rovers ein Gestein befindet, so soll er ein Feld vorwärtsfahren. Ansonsten soll er nichts tun.



2. a)
```
if(huegelVorhanden("rechts"))
{
  drehe("rechts");
}
```
Der else-Teil kann weggelassen werden.

b)
```
if(huegelVorhanden("rechts"))
{
  drehe("rechts");
}
else
{
  if(huegelVorhanden("links"))
  {
      drehe("links");
  }
  else
  {
      setzeMarke();
  }
}
```

SB S. 50

c)
```
if(markeVorhanden())
{
  entferneMarke();
}
else
{
  setzeMarke();
}
```

3. a) Pseudocode
```
solange keine Marke vorhanden
wiederhole
  wenn Hügel links und Gestein vorhanden
      dann analysiere Gestein
  ende wenn
  fahre vorwärts
ende wiederhole
```

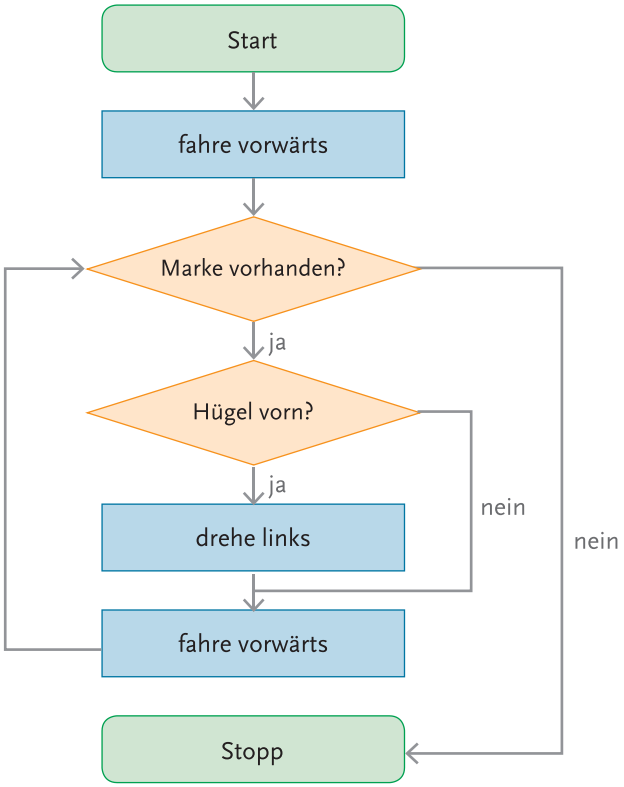
b)
```
public void methode3_4_3b()
{
  while(!markeVorhanden())
  {
      if(huegelVorhanden("links"))
      {
          if(gesteinVorhanden())
          {
              analysiereGestein();
          }
      }
      fahre();
  }
}
```

c) Annahme: Auf dem Startfeld befindet sich kein Gestein.

```
public void methode3_4_3c()
{
  setzeMarke();
  fahre();
  while(!markeVorhanden())
  {
      if(huegelVorhanden("links"))
      {
          if(gesteinVorhanden())
          {
              analysiereGestein();
          }
      }
      fahre();
  }
  drehe("rechts");
  drehe("rechts");
  fahre();
  while(!markeVorhanden())
  {
    fahre();
  }
}
```

Falls Aufgabe 6 (S. 43) bearbeitet wurde, bietet sich hier auch der Einsatz einer nachprüfenden Wiederholung an. Der erste *fahre*-Befehl kann dann entfallen.

4.




```
public void methode3_4_4()  // Spur folgen
  {
      fahre();
      while(markeVorhanden())
      {
          if(huegelVorhanden("vorne"))
          {
              drehe("links");
          }
          fahre();
      }
  }
```

Falls Aufgabe 6 (S. 43) bearbeitet wurde, bietet sich hier auch der Einsatz einer nachprüfenden Wiederholung an. Der erste *fahre*-Befehl kann dann entfallen.

SB S. 51

5. a)

```
solange kein Gestein vorhanden
wiederhole
  wenn Hügel vorne
      dann umfahre Hügel
      sonst fahre vorwärts
  ende wenn
  analysiere Gestein
ende wiederhole
```

```
public void erkunde()
{
  while(!gesteinVorhanden())
  {
      if(huegelVorhanden("vorne"))
      {
          umfahreHuegel();
      }
      else fahre();
```

```
  }
  analysiereGestein();
}

public void umfahreHuegel()
{
  drehe("links");
  fahre();
  drehe("rechts");
  fahre();
  fahre();
  drehe("rechts");
  fahre();
  drehe("links");
}
```

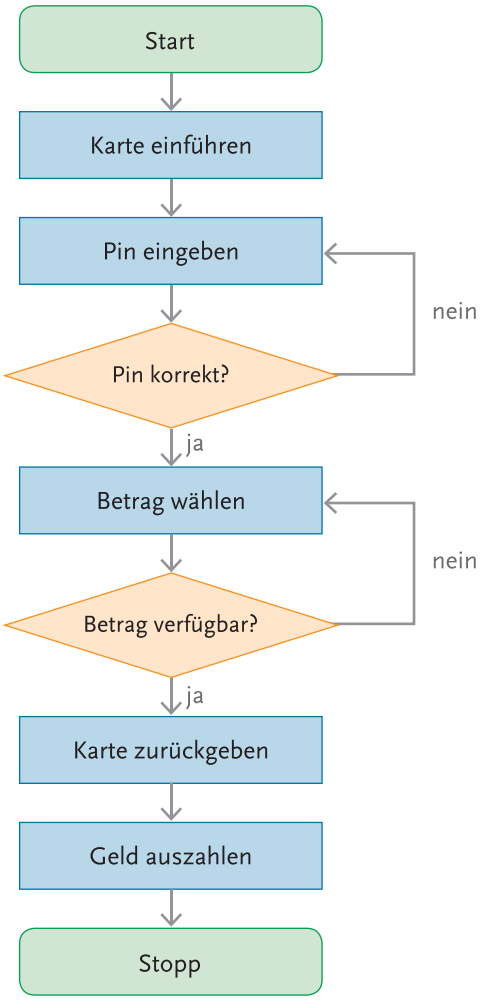
b) Die Methode *umfahreHuegel()* muss angepasst werden.

```
public void umfahreHuegel()
{
  drehe("links");
  fahre();
  drehe("rechts");
  fahre();
  while(huegelVorhanden("rechts"))
  {
      fahre();
  }
  drehe("rechts");
  fahre();
  drehe("links");
}
```

c) Die Methode *erkunde()* muss angepasst werden.

```
public void erkunde()
{
  setzeMarke();
  while(!gesteinVorhanden())
  {
      if(huegelVorhanden("vorne"))
      {
          umfahreHuegel();
      }
      else fahre();
  }
  analysiereGestein();
  drehe("rechts");
  drehe("rechts");
  while(!markeVorhanden())
  {
      if (huegelVorhanden ("vorne"))
      {
          umfahreHuegel ();
      }
      else fahre();
  }
}
```

6. Ohne die bedingte Anweisung wäre es nicht möglich, aufgrund von aktuellen Werten Entscheidungen in einem Programm zu treffen.
So muss ein Geldautomat beispielsweise prüfen, ob die eingegebene PIN korrekt ist oder nicht. Je nachdem, wie diese Prüfung ausfällt, muss der Automat sich anders „verhalten".
Ein vereinfachter Ablauf des Vorgangs „Geld abheben" als PAP (z. B. wurde nicht berücksichtigt, dass nur drei Versuche bei der PIN-Eingabe bestehen):



7. **a)**

```
wenn Tor Heim
    dann wiederhole 5 mal
        TOR anzeigen
    ende wiederhole
ende wenn
wenn Tor Gast
    dann wiederhole 3 mal
        TOR anzeigen
    ende wiederhole
ende wenn
```

b)

```
wenn Tor Heim
    dann TOR anzeigen
    Figur anzeigen
    wiederhole 4 mal
        TOR anzeigen
ende wiederhole
```

```
ende wenn
wenn Tor Gast
    dann TOR anzeigen
    Figur anzeigen
    wiederhole 2 mal
        TOR anzeigen
    ende wiederhole
ende wenn
```

SB S. 52

8. a) Der Zähler i nimmt die Werte 1, 3, 5, 7 und 9 an. Die Schleife wird also fünfmal durchlaufen.

b) 1
Hat nicht geklappt.
3
Hat nicht geklappt.
5
Hat nicht geklappt.
7
Hat nicht geklappt.
9
Hat nicht geklappt.

c) z. B. `if (i%2 != 0), if (!(i%2 == 0)), if (i%2 < 0), if (i%2 > 0)`

9. *If* in anderen Programmiersprachen

a) Falls der Benutzer Karl heißt, wird die Frage „Bist du Checker?“ ausgegeben.

b) Falls der Benutzer Karl heißt, wird die Frage „Bist du Checker?“ ausgegeben. Ansonsten, falls der Benutzer Andrea heißt, wird die Frage „Bist du Knolle?“ ausgegeben. Falls keine der beiden Bedingungen zutrifft, wird der String „Also du bist sicher weder Checker noch Knolle!“ ausgegeben.

SB S. 53

c)

a	b	Ergebnis
4	2	a größer gleich b
5	11	a kleiner b
7	7	a größer gleich b
323	233	a größer gleich b

d)

number	digits
17	2
5	1
3	1
99	2
899	3

3.5 Logische Operatoren

SB S. 57

1. a) Fall 1: Das Gestein wird aufgenommen, der Rover fährt ein Feld vorwärts und legt eine Marke ab.
Fall 2: Das Gestein wird aufgenommen und der Rover legt eine Marke ab.
Fall 3: Der Rover legt eine Marke ab.