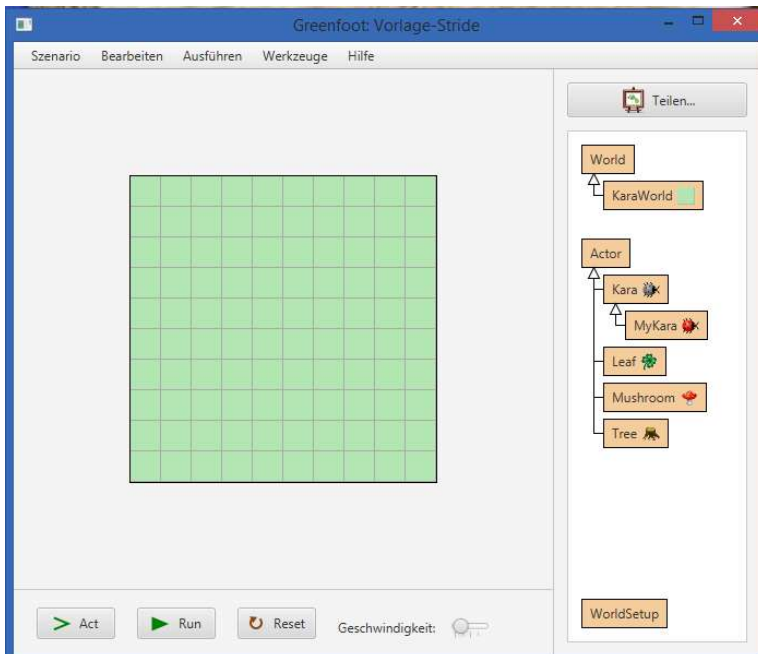


L1_1.1 Die Programmumgebung Greenfoot Stride

Greenfoot ist eine interaktive Entwicklungsumgebung, die an der Universität Kent entwickelt wurde. Sie ermöglicht eine einfache und spielerische Einführung in die Programmierung.



Sobald Greenfoot gestartet und das Szenario "Vorlage-Stride" geöffnet wurde, erscheint die abgebildete Bildschirmoberfläche.

Das Szenario "Vorlage-Stride" enthält bereits eine 10 x 10 Felder große „Spielfläche“ - die Welt - und stellt verschiedene Akteure zur Verfügung.

In der Programmierungsumgebung Greenfoot mit Stride interagiert der Käfer MyKara mit Akteuren (Blatt, Pilz und Baum) in einer Welt aus Quadraten.

Wichtiger Hinweis: Das Szenario "Vorlage-Stride" dient als Grundlage mehrerer Aufgabenstellungen der vorliegenden Unterrichtsmaterialien und enthält noch keinen Programmcode. Bevor Sie mit der Bearbeitung der einzelnen Aufgabenstellungen beginnen, muss das Szenario unter neuem Namen gespeichert werden.

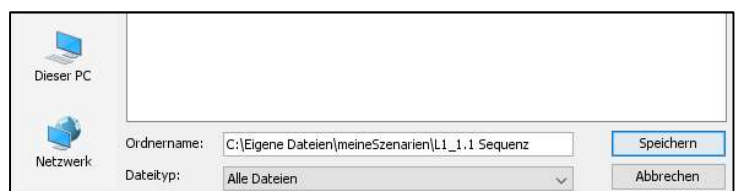
1 Szenario speichern

Beim Speichern eines in Greenfoot entwickelten Szenarios muss unterschieden werden, ob nur der eingefügte Programmcode oder auch die in der Kara-Welt platzierten Akteure gespeichert werden sollen.

Speichern des Programmcodes

► Befehlsfolge Szenario → Speichern als...

Hierbei wird in dem Szenario, das als Grundlage der Bearbeitung geöffnet wurde, und der entwickelte Programmcode unter einem neuen Ordernamen gespeichert.



Veränderungen in der Kara-Welt werden dagegen nicht gespeichert. Das heißt, dass alle Akteure (Käfer, Baum etc.), die auf der Welt platziert wurden, beim erneuten Öffnen des Szenarios nicht mehr vorhanden sind.

Speichern der der Kara-Welt

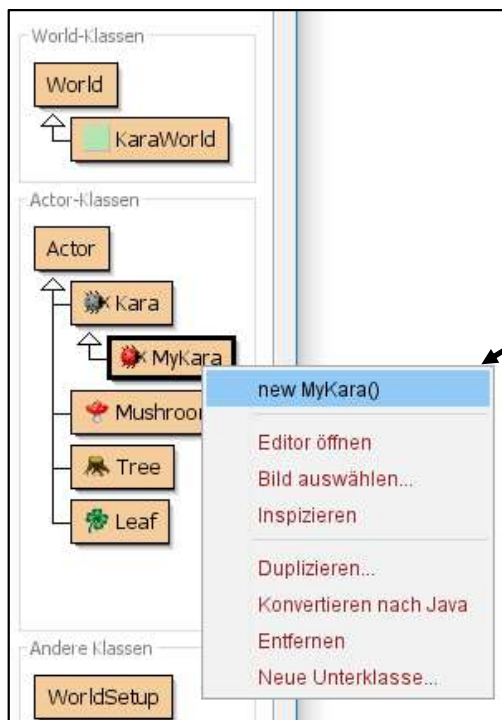
► Befehlsfolge **Ausführen** → **Die Welt speichern**

Hierbei werden in dem Szenario, das als Grundlage der Bearbeitung geöffnet wurde, sowohl der entwickelte Programmcode als auch alle Änderungen an der Kara-Welt gespeichert. Das heißt, dass alle Akteure (Käfer, Baum etc.), die auf der Welt platziert wurden, beim erneuten Öffnen des Szenarios wieder vorhanden sind.

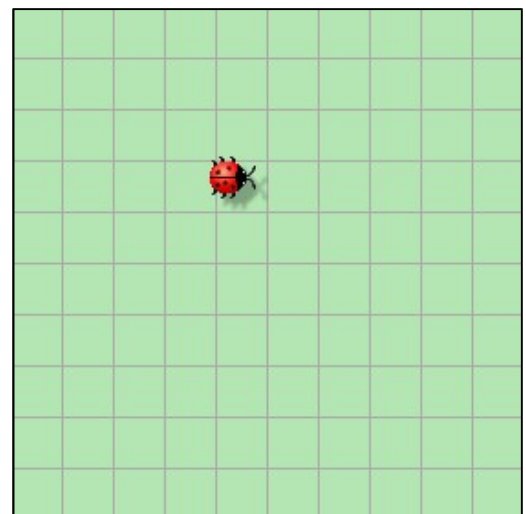
Für die Dokumentation der im Unterricht entwickelten Problemlösungen empfiehlt sich somit folgende Vorgehensweise:

- Zunächst das Szenario mit neuem Namen speichern (Szenario → Speichern als...). Damit wird der entwickelte Programmcode gespeichert.
- Anschließend dieses Szenario mit den verwendeten Akteuren und dem entwickelten Programmcode speichern (Ausführen → Die Welt speichern).

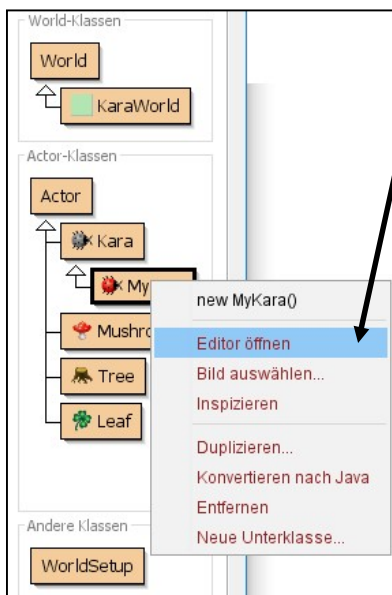
2 Akteure verwenden und Aktionen zuweisen



Neue Akteure werden mit Hilfe des Kontextmenüs (rechte Maustaste) aufgerufen (hier: „newMyKara()“).

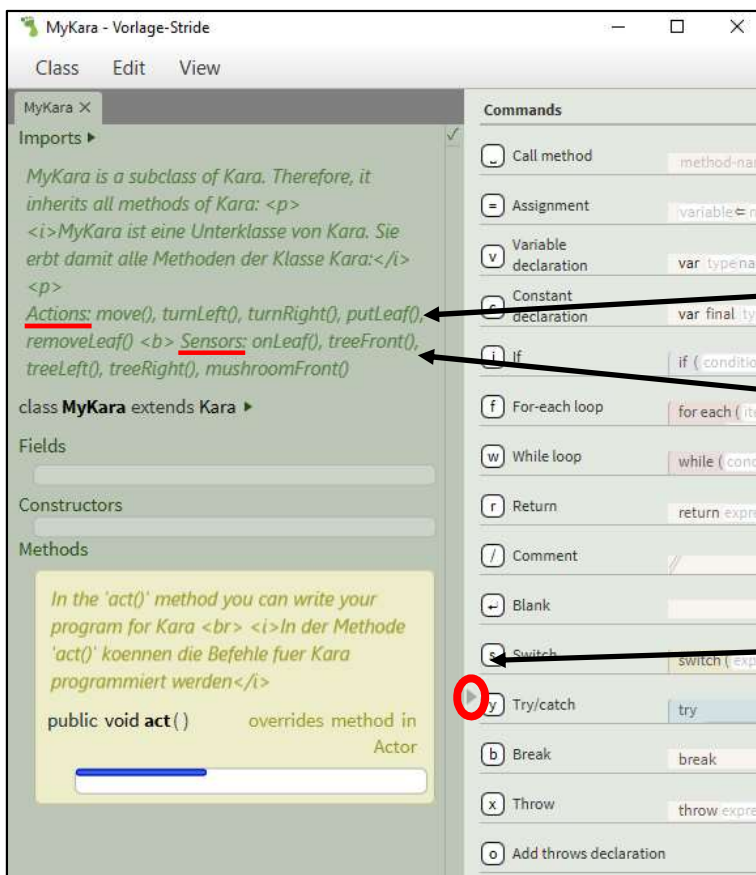


Anschließend wird der Akteur mit Hilfe der Maus auf einem beliebigen Feld in der Welt platziert.



Um MyKara zu „programmieren“, wird über das Kontextmenü der Editor geöffnet (Editor öffnen).

Hier werden die einzelnen Aktionen festgelegt, die während des Programmablaufs „abgearbeitet“ werden sollen. Man spricht hierbei von der Kodierung des Programms.



Das Editor-Fenster lässt sich in zwei Bereiche unterteilen:

Auf der linken Seite (dunkelgrün unterlegt) werden die möglichen Aktionen und Sensoren von MyKara genannt, die in dieser Stride-Umgebung zur Verfügung stehen.

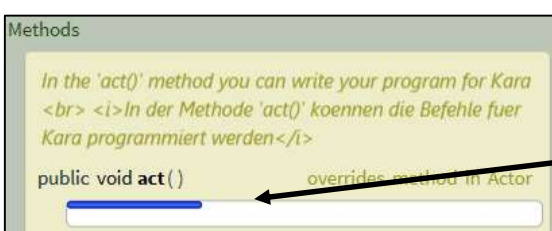
Actions: move(), turnLeft(), turnRight(), putLeaf(), removeLeaf()

Sensors: onLeaf(), treeFront(), treeLeft(), treeRight(), mushroomFront()

Auf der rechten Seite ist ein "Spickzettel" zu sehen, der die Programmelemente anzeigt, die eingesetzt werden können.

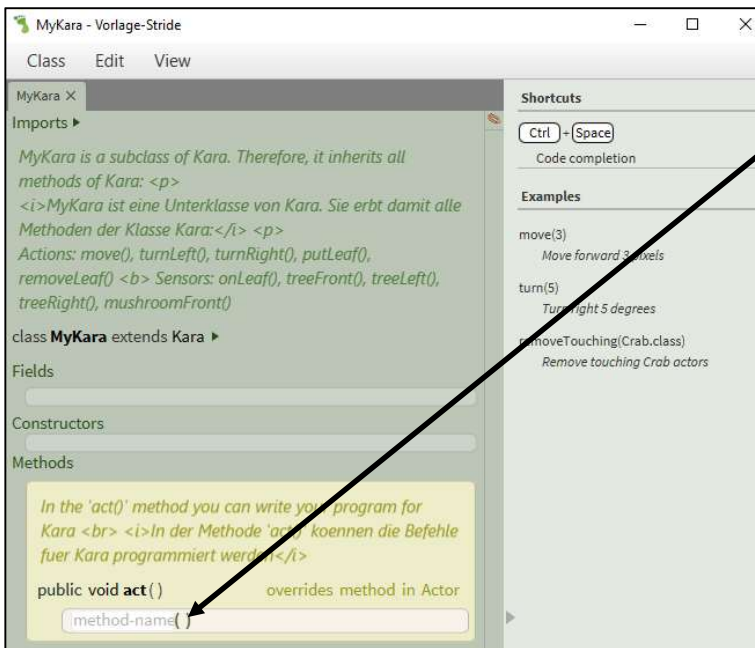
Hierzu zählen u.a. die Variablendeklaration (Variable declaration), die for-Schleife (For-each loop), die while-Schleife (While loop) und die if-Verzweigung (if).

Dieses Fenster kann ein- bzw. ausgeblendet werden.



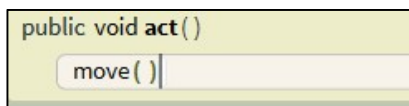
Die Standardmethode `act()` wird benutzt, um MyKara Aktionen ausführen zu lassen.

Die Eingabezeile zeigt einen blauen Balken, da noch kein Eintrag vorhanden ist.



Ist der Cursor im Methodenkörper, weicht der blaue Balken durch **Drücken der Leertaste** einer Eingabezeile.

Der Spickzettel ändert sich ebenfalls, und entsprechende Hinweise erscheinen. So wird hier beispielsweise die Aktion `move()` genannt und mit einem Beispiel erläutert.



Wird hier nun eine solche Aktion eingetragen, ist dies eine Anweisung für MyKara, etwas zu tun (hier: einen Schritt vorwärts gehen).

Das Editor-Fenster zeigt mit einem kleinen Symbol in der oberen rechten Ecke an, ob Sie sich im Eingabemodus befinden, ob Ihre eingegebene Anweisung fehlerhaft bzw. unvollständig ist oder ob die Anweisung syntaktisch korrekt ist.



Der Stift, während man Anweisungen editiert.



Das rote Kreuz, solange der Programmcode fehlerhaft oder unvollständig ist.



Der grüne Haken, wenn der Code syntaktisch richtig ist.



Falls sich nach der Eingabe im Editor die Welt in Grautönen und verschwommen zeigt und der manipulierte Akteur schraffiert ist, bedeutet dies, dass das Programm noch nicht übersetzt (kompiliert) wurde.



Prüfen Sie, ob der grüne Haken vorhanden ist bzw. korrigieren Sie eventuell in der Codierung vorhandene Fehler.

Folgende Übersicht beschreibt die verschiedenen Möglichkeiten, mit denen der Käfer MyKara ausgestattet werden kann:

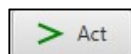
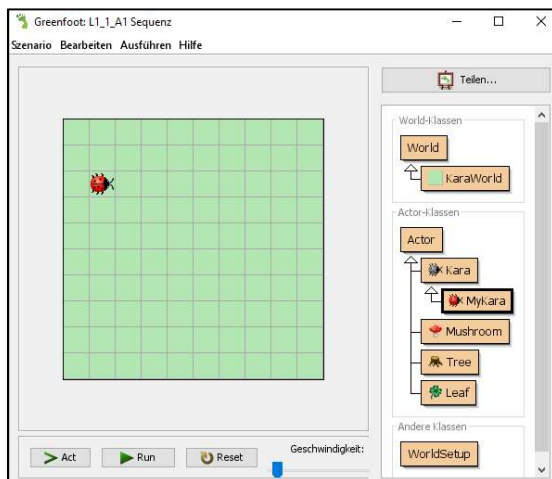
Aktionen (Anweisungen, die ausgeführt werden)

move()	MyKara macht einen Schritt in die aktuelle Richtung.
turnLeft()	MyKara dreht sich um 90° nach links.
turnRight()	MyKara dreht sich um 90° nach rechts.
putLeaf()	MyKara legt ein Kleeblatt an die Position, auf der er sich befindet.
removeLeaf()	MyKara entfernt ein unter ihm liegendes Kleeblatt.

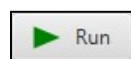
Sensoren (Bedingungen, die geprüft werden)

onLeaf()	MyKara schaut nach, ob er sich auf einem Kleeblatt befindet.
treeFront()	MyKara schaut nach, ob sich ein Baum vor ihm befindet.
treeLeft()	MyKara schaut nach, ob sich ein Baum links von ihm befindet.
treeRight()	MyKara schaut nach, ob sich ein Baum rechts von ihm befindet.
mushroomFront()	MyKara schaut nach, ob er einen Pilz vor sich hat.

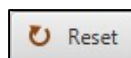
Nachdem die gewünschten Aktionen im Editor-Fenster festgelegt wurden, kann der Käfer mit Hilfe der Maus auf einem beliebigen Feld in der Welt platziert werden und mit der Schaltfläche



die Methode *act()* einmal aufgerufen,



die Methode *act()* immer wiederaufrufen,



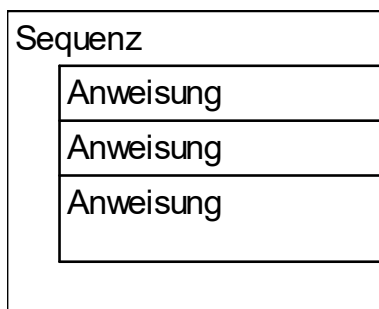
die Welt komplett zurückgesetzt werden, wobei Ihre Programmierung erhalten bleibt.

L1_1.2 Die Sequenz (Linearer Ablauf)

Die Entwicklung von Problemlösungen mit Hilfe einer Software erfolgt in der Regel in zwei Schritten:

- ▶ der Strukturierung der einzelnen Arbeitsschritte (Struktogramm),
- ▶ der Übersetzung der Arbeitsschritte in eine Programmiersprache (Programmcode).

Struktogramm



Jede einzelne Anweisung wird in einen rechteckigen Strukturblock geschrieben. Die Strukturblöcke werden nacheinander von oben nach unten durchlaufen.

Leere Strukturblöcke sind nur in Verzweigungen zulässig.

Programmcode

Eine Anweisungssequenz ist eine Folge von Anweisungen, die der Reihe nach ausgeführt werden. Kara-Programme bestehen in der Regel aus vielen Anweisungen.

```

MyKara X
<i>MyKara ist eine Unterklasse von Kara. Sie erbt damit alle Methoden der Klasse Kara:</i> <p> ^✓
Actions: move(), turnLeft(), turnRight(), putLeaf(), removeLeaf() <b> Sensors: onLeaf(), treeFront(),
treeLeft(), treeRight(), mushroomFront()

class MyKara extends Kara ▶

Fields
Constructors
Methods

In the 'act()' method you can write your program for Kara <br> <i>In der Methode 'act()'
koennen die Befehle fuer Kara programmiert werden</i>

public void act() overrides method in Actor
{
    move()
    move()
    turnRight()
    move()
    turnLeft()
    move()
}

```

L1 1.1 L1_2 Die Zählerschleife (for-Schleife)

Beim Programmieren kommt es immer wieder vor, dass ein oder mehrere bestimmte Schritte mehrfach hintereinander ausgeführt, also wiederholt werden müssen. Das würde z.B. bei 1000 oder 100.000 nötigen Wiederholungen zu einer endlosen Kette identischer Anweisungen führen bzw. wäre vielleicht auch gar nicht mit vertretbarem Aufwand umsetzbar.

Zur Arbeitserleichterung gibt es daher für solche Wiederholungen spezielle Anweisungen. Statt nun z.B. 10 oder 20 Mal hintereinander den gleichen Befehl zu programmieren, kann man angeben:

„Führe diese Anweisung(en) 14 Mal aus“.

Erreicht wird das mit einer so genannten „Zählerschleife“ oder auch „for-Schleife“.

5.1.1 1 Aufbau einer Zählerschleife

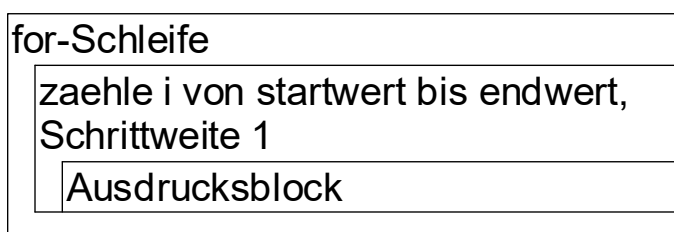
Eine Zählerschleife benötigt drei Elemente:

1. Einen Zähler, mit dem die Wiederholungen gezählt werden.
2. Mit welcher Zahl soll die Zählung beginnen?
3. Bis wohin soll gezählt werden?

Für 14 Wiederholungen könnte z.B. mit der Zahl 1 begonnen und bis zur Zahl 14 gezählt werden. Diese Informationen stehen im so genannten „**Schleifenkopf**“.

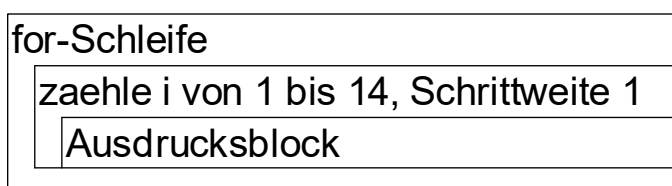
Im „**Schleifenkörper**“, der direkt auf den **Schleifenkopf** folgt, stehen dann genau die Anweisungen, die bei jedem Schleifendurchlauf ausgeführt werden sollen, in unserem Beispiel z.B. "Kara, mache einen Schritt vorwärts" → *move()*.

5.1.2 2 Struktogramm einer Zählerschleife



Beispiel:

Für eine 14-malige Wiederholung von Programmanweisungen muss die Schleifenbedingung folgendermaßen formuliert werden:



5.1.3 3 Syntax einer Zählerschleife in Greenfoot Stride

Für die Kodierung von Zählerschleifen wird in Greenfoot Stride die *for-each-Schleife* verwendet.

for each(Datentyp des Zählers Bezeichnung des Zählers in Startwert .. Endwert)

Schleifenkopf:

```
public void act()      overrides method in Actor
{
    for each (int zaehler in 1 .. 14)
    {
        move();
    }
}
```

Im Schleifenkopf wird der Zähler vereinbart und erhält einen Startwert.

Der Zähler wird bis zum Ende des angegebenen Bereichs hochgezählt.

In Greenfoot Stride beträgt die Schrittweite der *for-each-Schleife* 1 und muss nicht gesondert angeführt werden.

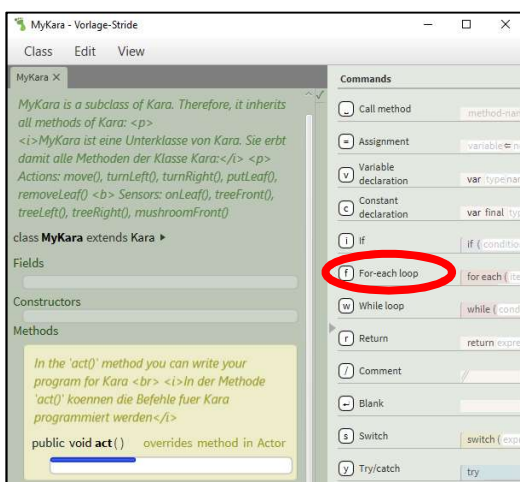
Hinweis: Da der Zähler eine Ganzzahl ist, wird in Greenfoot der Zähler als Variable vom Typ Integer (int) festgelegt.

Im dargestellten Beispiel startet die Zählvariable (*zaehler*) mit dem Wert 1. Nach jeder Erhöhung der Zählervariablen wird die Schleife durchlaufen. Nachdem der Zähler den Wert 14 erreicht hat, erfolgt der letzte Wiederholungsvorgang.

Schleifenkörper:

Im Schleifenkörper wird die Anweisung angegeben, die bei jeder Wiederholung ausgeführt werden soll. Hier ist es die Anweisung, dass MyKara einen Schritt vorwärts machen soll, also einfach *move()*.

5.1.4 4 Eingabe des Schleifenkopfes im MyKara-Editor



Um den Schleifenkopf in den Methodenkörper der Methode *act()* einzugeben, muss die rechte Seite des Editor-Fensters geöffnet sein. Hier befinden sich die Auswahlmöglichkeiten für verschiedene Kommandos.

Für den Schleifenkopf der *for*-Schleife ist der Eintrag *for-each-loop* per Mausklick zu wählen.

Im Methodenkörper der Methode *act()* erscheint daraufhin die unvollständige 'for each'-Anweisung.

```
public void act()      overrides method in Actor
{
    for each (item type:item name in
    collection)
    {
    }
}
```

Sie ist entsprechend der Aufgabenstellung zu ergänzen ist.

```
public void act()      overrides method in Actor
{
    for each (int zahler in 1 .. 9)
    {
    }
}
```


L1_3 Die while-Schleife

Die „while“-Schleife löst Problemstellungen, bei denen ein Vorgang solange wiederholt wird, bis ein Bedingung nicht mehr erfüllt ist.

Ein Beispiel wäre die Fragestellung: „Erhöhe den Wert der Variable *anzahl* solange um 1 und gebe sie aus, solange sie den Wert 10 nicht erreicht hat“.

5.1.5 1 Vergleichsoperatoren in Greenfoot

Vergleichsoperatoren werden immer zum Vergleich zweier Werte benutzt und finden ihre Anwendung zumeist beim Einsatz von Wiederholungen oder Alternativen. Als Ergebnis des Vergleichs wird ein boolescher Wert (true / false) zurückgegeben. Meist wird eine Variable mit einem festen Wert oder eine Variable mit einer anderen Variablen verglichen. Je nach Ergebnis des Vergleichs wird das Programm eine andere Reaktion zeigen.

Operator	Beispiel	Beschreibung
<	a < b	a ist kleiner als b
<=	a <= b	a ist kleiner oder gleich b
>	a > b	a ist größer als b
>=	a >= b	a ist größer oder gleich b
==	a == b	a ist gleich b
!=	a != b	a ist ungleich b

5.1.6 2 Logische Operatoren in Greenfoot

MyKara kann durch die Nutzung seiner Sensoren auf Ereignisse reagieren. MyKara ist es auch möglich, auf mehrere Sensoren gleichzeitig zu reagieren. Hier eine kurze Übersicht über die Verwendung von Operatoren.

Operator	Beispiel	Beschreibung	Erläuterung
&&	treeFront() && treeRight()	und	Die Bedingung ist erfüllt (true), wenn beide Aussagen zutreffen, d.h. wenn MyKara einen Baum vor und rechts von sich hat.
^(*)	treeFront() treeRight()	oder	Die Bedingung ist erfüllt (true), wenn die eine oder die andere oder beide Aussage/n erfüllt sind.
!	!treeFront()	nicht	Das Ausrufezeichen ändert einen Ausdruck von true in false und umgekehrt. Hier ist die Bedingung erfüllt, wenn Kara nicht vor einem Baum steht.

*) Das pipe-Symbol | kann auf der Windows-Tastatur mit den Tasten *AltGr* + < erzeugt werden.

5.1.7 3 Aufbau einer while-Schleife

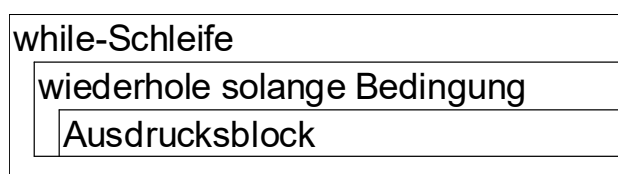
Die while-Schleife benötigt zuerst eine Bedingung, die dafür sorgt, dass die Schleife durchlaufen wird oder nicht. Diese wird im Schleifenkopf hinter dem Schlüsselwort *while* in Klammern angegeben.

Beispiel: `while (anzahl<10)`

Danach folgt der Schleifenkörper. Hier stehen die Anweisungen, die abgearbeitet werden sollen, falls es zu einem Schleifendurchlauf kommt.

Im Schleifenkörper stehen die Anweisungen, die bei jedem Schleifendurchlauf ausgeführt werden.

5.1.8 4 Das Struktogramm einer while-Schleife



Wiederholungsstruktur mit vorausgehender Bedingungsprüfung. Der Schleifenkörper (Anweisungsblock) wird nur durchlaufen, solange die Bedingung erfüllt (wahr) ist.

5.1.9 5 Syntax einer while-Schleife

```
while ( anzahl < 10 )
{
  move ( )
  anzahl ← anzahl + 1
}
```

Im **Schleifenkopf** wird die Bedingung festgelegt.

Der **Schleifenkörper** enthält die Anweisungen, die bei jeder Wiederholung ausgeführt werden.

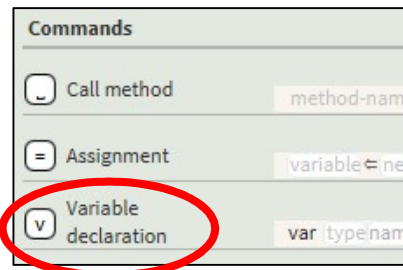
Im dargestellten Beispiel wird geprüft, ob der Wert der Variablen *anzahl* < 10 ist.

Ist dies der Fall, wird der Schleifenkörper abgearbeitet:

Der Wert der Variablen *anzahl* wird um 1 erhöht.

Danach wird wieder geprüft, ob der Wert der Variablen *anzahl* immer noch kleiner als 10 ist ...

5.1.10 6 Eingabe des Programmcodes im MyKara-Editor

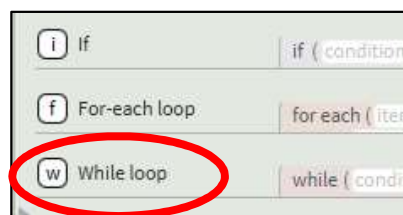


Zur Deklaration der Variable *anzahl* muss der Eintrag *Variable declaration* auf der rechten Seite des Editor-Fensters per Mausklick gewählt

```
public void act() overrides method in Actor
    var typename ← value
```

und entsprechend der Aufgabenstellung ergänzt werden.

```
public void act() overrides method in Actor
    var int anzahl ← 0
```

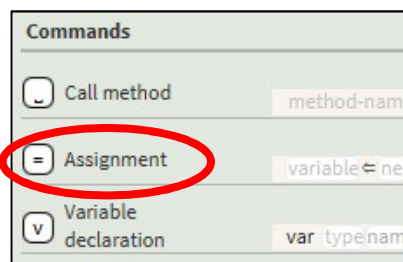


Für die Anweisung des Schleifenkopfs ist der Eintrag *While loop* zu wählen

```
public void act() overrides method in Actor
    var int anzahl ← 0
    while (condition)
```

und entsprechend der Aufgabenstellung zu ergänzen.

```
while (anzahl < 10)
```



Für die Erhöhung des Variablenwerts wird der Eintrag *Assignment* gewählt.

```
public void act() overrides method in Actor
    var int anzahl ← 0
    while (anzahl < 10)
        variable ← new-value
```

und mit folgendem Eintrag ergänzt.

```
while (anzahl < 10)
    anzahl ← anzahl + 1
```

Nachdem auch die Aktion *move()* in den Methodenrumpf der Methode *act()* eingefügt wurde, ergibt sich folgender Programmcode:

```
public void act() overrides method in Actor
    var int anzahl ← 0
    while (anzahl < 10)
        move()
        anzahl ← anzahl + 1
```

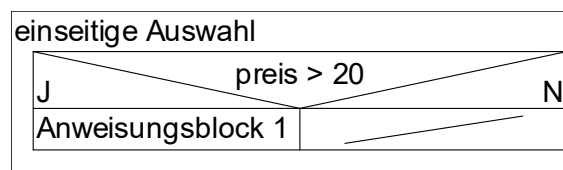
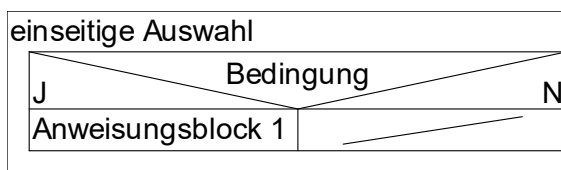
L1_4 Die Alternative

Häufig soll ein Programm etwas ausführen, das von einer aktuellen Situation abhängig ist. Beispielsweise könnte MyKara prüfen, ob er auf einem Blatt steht und dieses aufheben, falls er tatsächlich auf einem Blatt steht, oder einen Schritt vorwärts gehen, wenn er nicht auf einem Blatt steht.

Dieses Vorgehen wird als „Fallunterscheidung“ bezeichnet: In dem einen „Fall“ soll das Programm etwas anderes machen als in einem anderen „Fall“. Bekannt ist die Situation vom Einsatz der Wenn-Funktion in der Tabellenkalkulation.

In Stride wird eine Fallunterscheidung mit der Anweisung *if()* - *else* umgesetzt. Dabei steht in der Klammer hinter dem *if* eine Bedingung, die entweder erfüllt ist oder nicht. Danach folgt die Anweisung für den Fall, dass die Bedingung erfüllt ist. Sollte dies nicht der Fall sein, so wird die Anweisung der *else*-Alternative (so vorhanden) abgearbeitet.

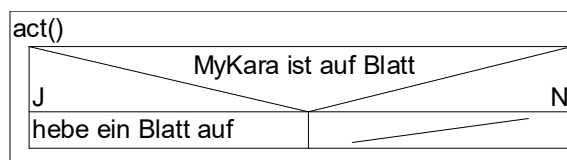
1.1 Die einseitige Auswahl (bedingte Verarbeitung)



Nur wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Trifft die Bedingung nicht zu (falsch), wird der Durchlauf ohne eine weitere Anweisung fortgeführt (Austritt unten).

1.2 Syntax einer einseitigen Auswahl

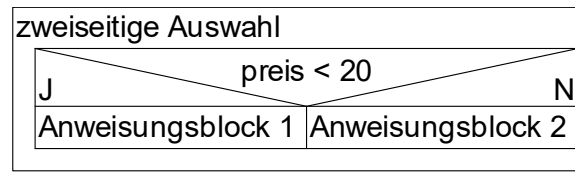
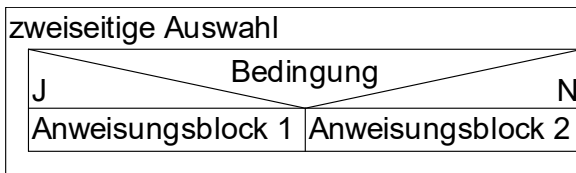
In der Bedingung wird unterstellt, dass nach der positiven Erfüllung gesucht wird. Auf die Angabe „ist wahr“ wird verzichtet.



Im dargestellten Beispiel wird die Aktion *removeLeaf()* ausgeführt, wenn die Bedingung *onLeaf()* eintritt. Ansonsten passiert nichts.

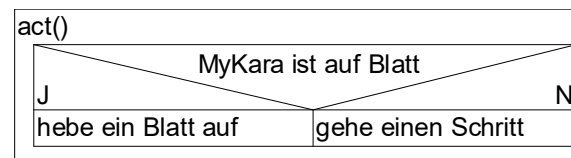
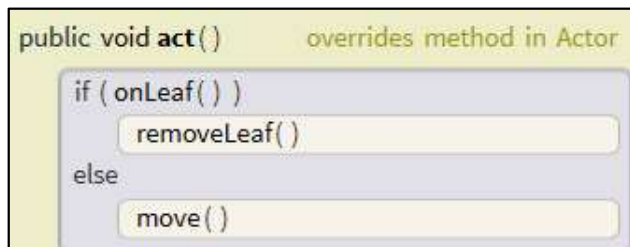
Das heißt, wenn ein Akteur auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Tritt dieser Fall ein, bewirkt die Aktion *removeLeaf()*, dass er das Blatt entfernt.

2.1 Die zweiseitige Auswahl (alternative Verarbeitung)



Wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Trifft die Bedingung nicht zu (falsch), wird der Anweisungsblock 2 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Der Verzweigungsblock wird nach der Ausführung des jeweiligen Anweisungsblocks verlassen

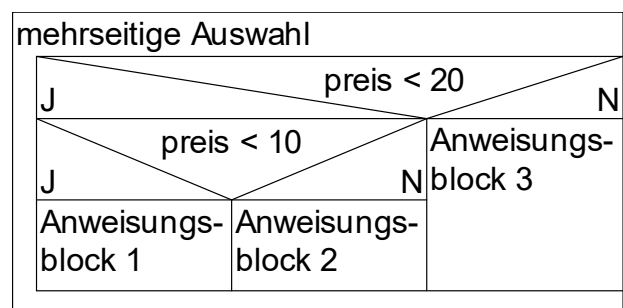
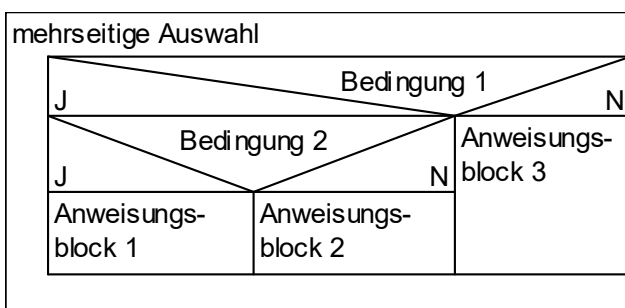
2.2 Syntax einer zweiseitigen Auswahl



Im dargestellten Beispiel wird die Aktion *removeLeaf()* ausgeführt, wenn die Bedingung eintritt. Ansonsten geht MyKara einen Schritt - *move()* - nach vorne.

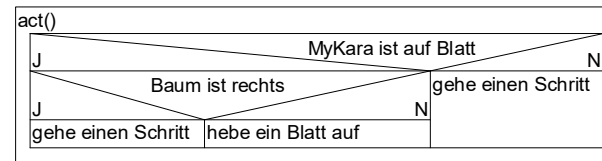
Das heißt, wenn ein Akteur auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Tritt dieser Fall ein, bewirkt die Aktion *removeLeaf()*, dass er das Blatt entfernt. Wenn der Akteur auf keinem Blatt steht, geht er einen Schritt vorwärts.

3.1 Die mehrseitige Auswahl



Auch „verschachtelte“ Auswahl genannt, da eine weitere Bedingung folgt. Die Verschachtelung ist ebenso im Nein-Fall möglich

3.2 Syntax einer mehrseitigen Auswahl



Im dargestellten Beispiel wird die Aktion *move()* ausgeführt, wenn beide Bedingung eingetreten sind. Ist nur die erste, nicht aber die zweite Bedingung erfüllt, wird die Aktion *removeLeaf()* ausgeführt. Wenn die erste Bedingung nicht erfüllt ist, wird die Aktion *move()* ausgeführt.

Das heißt, wenn ein Akteur auf einem Blatt steht, liefert der Sensor *onLeaf()* den Wert *wahr* zurück. Ist dies der Fall wird zusätzlich geprüft, ob ein Baum rechts von ihm steht. Trifft auch das zu, liefert der Sensor *treeRight()* den Wert *wahr* zurück. Die Aktion *move()* bewirkt, dass er einen Schritt geradeaus geht.

Liefert der Sensor *treeRight()* den Wert *false*, bewirkt die Aktion *removeLeaf()*, dass er das Blatt entfernt.

Wenn der Sensor *onLeaf()* den Wert *false* zurückgibt, führt die Aktion *move()* dazu, dass der Akteur einen Schritt vorwärts geht.

4 Eingabe des Programmcodes im MyKara-Editor



Um eine if-Verzweigung im Methodenrumpf der Methode *act()* eintragen zu können, ist der Eintrag *if* aus der rechten Seite des Editor-Fensters per Mausklick zu wählen.

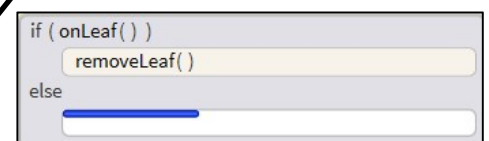
Im Methodenkörper der Methode *act()* erscheint daraufhin die unvollständige Kopfzeile der if-Anweisung,



die entsprechend der Aufgabenstellung zu ergänzen ist.



Sind auch Anweisungen für einen Alternativ-Fall (*else*) zu erfassen, muss an der markierten Stelle der Buchstabe 'e' (für *else*) getippt werden.



J1	BPE 5: Grundlagen der Programmierung Informationsmaterial	Informatik
----	---	------------

Anschließend werden in bekannter Weise die weiteren Aktionen erfasst.

```
public void act() overrides method in Actor
{
    if ( onLeaf() )
        removeLeaf()
    else
        move()
}
```