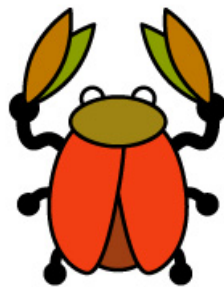


Bugs

Anleitung



Inhaltsverzeichnis

INHALTSVERZEICHNIS	2
WAS IST BUGS?	3
WIE STARTET MAN BUGS ?	3
AUFBAU DER OBERFLÄCHE	4
DIE MENÜLEISTE	5
DIE STEUERLEISTE	7
DIE KÄFERAUSWAHL	8
<i>Scarabeus simplicus – Käfer</i>	8
<i>Scarabeus laboranticus - Arbeiterkäfer</i>	8
<i>Scarabeus transporticus - Transportkäfer</i>	8
<i>Scarabeus sensoricus - Fühlerkäfer</i>	8
DER CODE EDITOR.....	9
DIE AUSGABE-KONSOLE.....	9
DIE BEFEHLSLEISTE	10
<i>Die Grundbefehle</i>	10
<i>Spezielle Befehle der einzelnen Käfer</i>	11
WAS IST EINE UMGEBUNG ?	12
<i>Rasen</i>	12
<i>Mauer</i>	12
<i>Kiste</i>	12
WIE ERSTELLT MAN EINE UMGEBUNG ?	13
WAS IST EIN SZENARIO ?	17
WIE ERSTELLT MAN EIN SZENARIO ?	17

Was ist Bugs?

Bugs ist ein Programm, welches dem Benutzer helfen soll, die *strukturierte Programmierung* zu erlernen, sowie erste Erfahrungen mit Java und *objektorientierter Programmierung* zu sammeln. Bugs wurde vollständig in Java geschrieben und läuft daher unter unterschiedlichen Betriebssystemen wie z.B. Windows oder Linux.

Aufgabe des Benutzers ist es, kleine Käfer mittels Java-Kommandos durch eine sog. *Umgebung* zu steuern.

Die Umgebung ist frei gestaltbar. Eine *Umgebung* kann Hindernisse wie Mauern oder andere Objekte wie z.B. Kisten enthalten. Weiterhin gibt es verschiedene Käfer mit unterschiedlichen Fähigkeiten.

Wie startet man Bugs ?

Um Bugs starten zu können benötigt man ein JRE (Java Runtime Environment) in der Version 1.5. Das JRE sorgt dafür, daß ein Java Programm ausgeführt werden kann. Es kann unter folgender Adresse heruntergeladen werden :

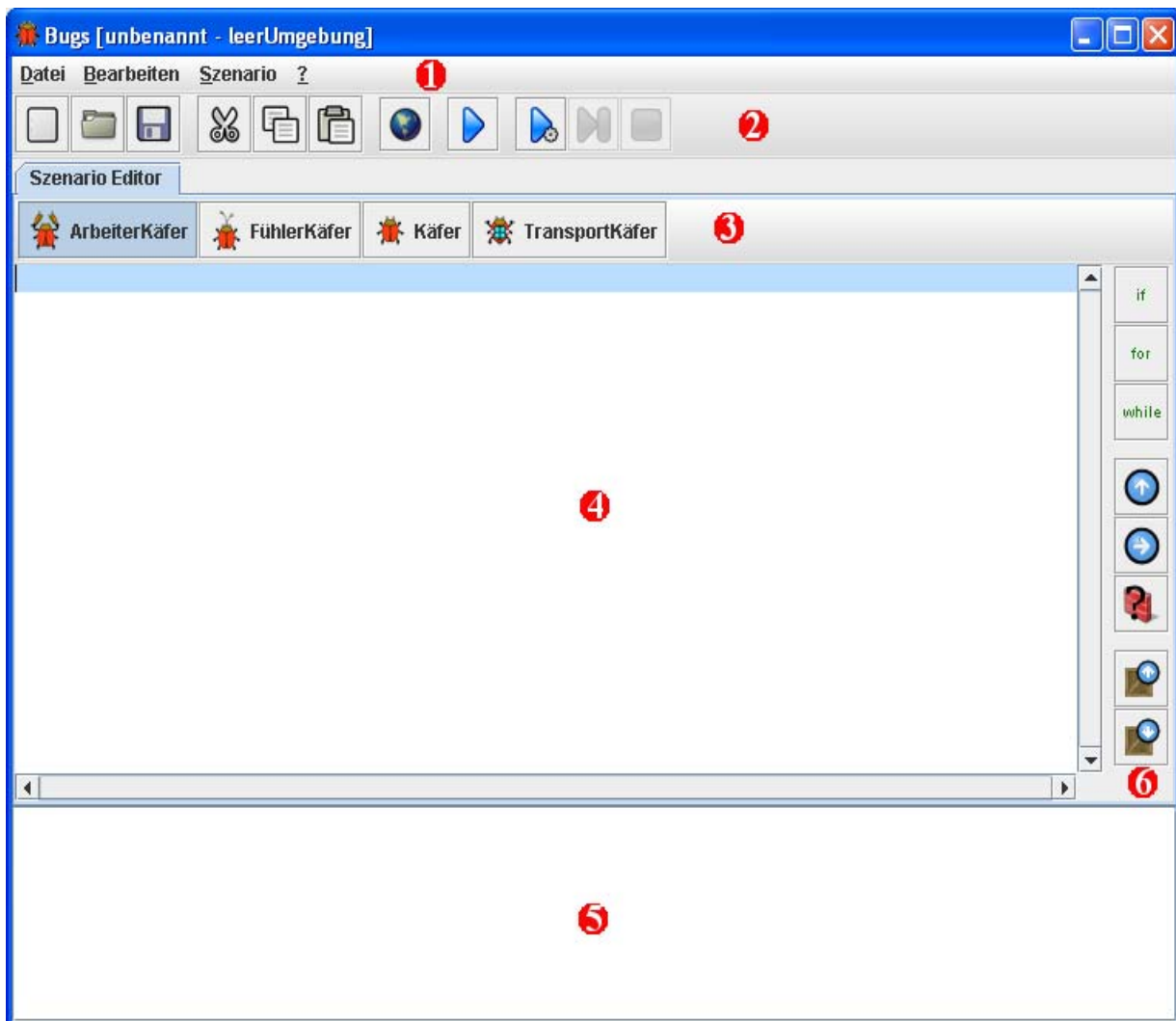
<http://java.sun.com/j2se/1.5.0/download.jsp>

Wenn ein JRE installiert ist, kopiert man den Ordner *Bugs* in ein Verzeichnis seiner Wahl.

Unter Windows reicht nun ein Doppelklick auf die Datei *start.bat* und Bugs startet .

Hat man alles richtig gemacht müsste das Haupt-Fenster von Bugs zu sehen sein.

Aufbau der Oberfläche



Das Haupt-Fenster von Bugs teilt sich auf in folgende Bereiche :

1. Die Menüleiste
2. Die Steuerleiste
3. Die Käferauswahl
4. Der Code-Editor
5. Die Ausgabe-Konsole
6. Die Befehlsleiste

Die Menüleiste

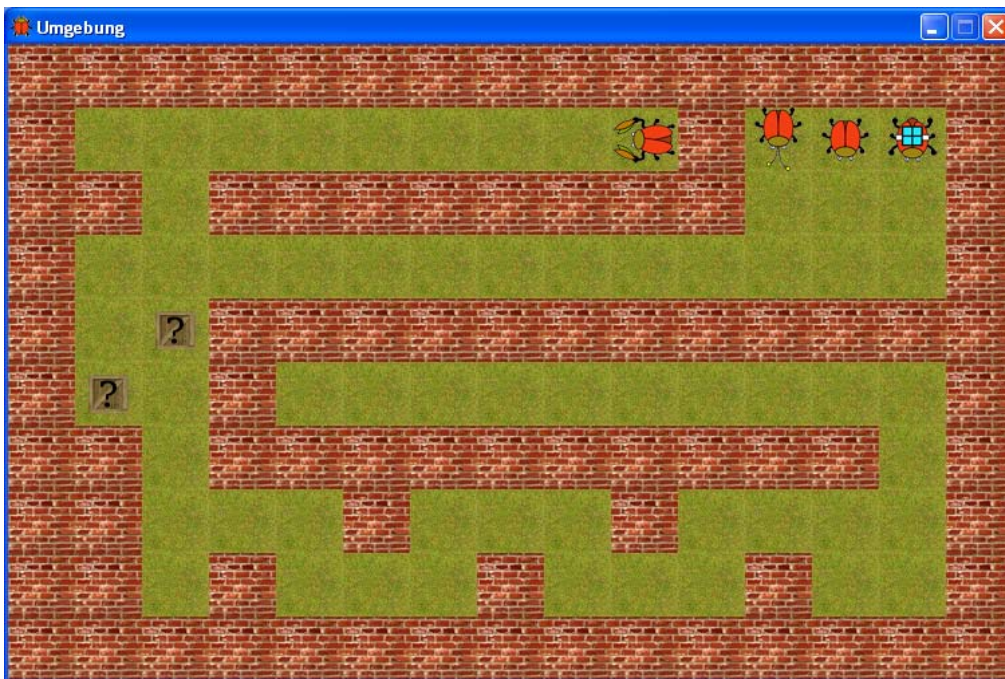
In der Menüleiste befinden sich folgende Menüs :

- Datei
- Bearbeiten
- Szenario
- ?

Unter dem Menüpunkt *Datei* hat man die Möglichkeit *Szenarien* zu öffnen, zu speichern und neu zu erzeugen. Was Szenarien sind wird unter Punkt 6 genauer erklärt. Es handelt sich hierbei einfach ausgedrückt um Bugs-Programme.

Der Menüpunkt *Bearbeiten* enthält Befehle um mit der Zwischenablage kommunizieren zu können oder das Editieren des Codes zu erleichtern.

Das Menü *Szenario* enthält folgende Befehle :



- *Umgebung anzeigen*
Klickt man hierauf öffnet sich ein zweites Fenster, welches die sog. *Umgebung* enthält.
- *Ausführen*
Dieser Befehl führt das Programm aus dem *Code-Editor* aus. Das Programm wird daraufhin kompiliert. Enthält es keine Fehler wird automatisch die *Umgebung* geöffnet und das Programm ausgeführt, andernfalls wird eine Fehlermeldung in der *Konsole* ausgegeben.
- *Debuggen*
Debuggen soll ermöglichen, dass man sein Programm schrittweise ausführen kann (Befehl für Befehl). Im wesentlichen passiert genau das, was passiert wenn man auf *Ausführen* klickt. Der Unterschied liegt nur darin, dass nur der erste Befehl des Programms im *Code-Editor* ausgeführt wird. Für jeden weiteren Befehl klickt man im Szenario-Menü auf *Schritt*. Die Zeile mit dem aktuellen Befehl, der ausgeführt werden soll, wird im *Code-Editor* orange unterlegt. Möchte man das *Debugging* beenden klickt

man auf Debuggen stoppen.

Das sog. *Debugging* befindet sich jedoch noch in einem experimentellem Stadium. Es kann auf einigen Computern zu undefiniertem Verhalten führen und sogar die JavaVM zum Absturz bringen. Diese Funktion sollte also nur mit Vorsicht genossen werden !

Die Steuerleiste

Die Steuerleiste enthält Buttons entsprechend zu den Menüpunkten der *Menüleiste* :



Neues *Szenario* erstellen



Szenario öffnen



Szenario speichern



Ausschneiden



Kopieren



Einfügen



Umgebung öffnen



Ausführen



Debuggen



Schritt



Debuggen stoppen

Die Käferauswahl



Hier lassen sich alle Käfer, die man in Bugs durch Programme steuern kann, auswählen. Beschrieben werden die verschiedenen Modelle hier:



Scarabeus simplicus – Käfer

Dieser Käfer hat nicht viele Fähigkeiten. Er kann einen Schritt nach vorne machen und sich nach rechts drehen. Auch ist er in der Lage festzustellen ob sich ihm etwas in den Weg gestellt hat.



Scarabeus laboranticus - Arbeiterkäfer

Dieses Exemplar ist ein paar Evolutionsschritte weiter. Er hat natürlich alle Fähigkeiten die sein Verwandter oben auch hat. Zusätzlich ist er aber in der Lage eine Kiste von einem Ort zum anderen zu tragen oder diese seinem Freund dem Transportkäfer auf den Rücken zu stellen. Die verschiedenen Arten von Kisten werden weiter unten in „4. Was ist eine Umgebung?“ erklärt.



Scarabeus transporticus - Transportkäfer

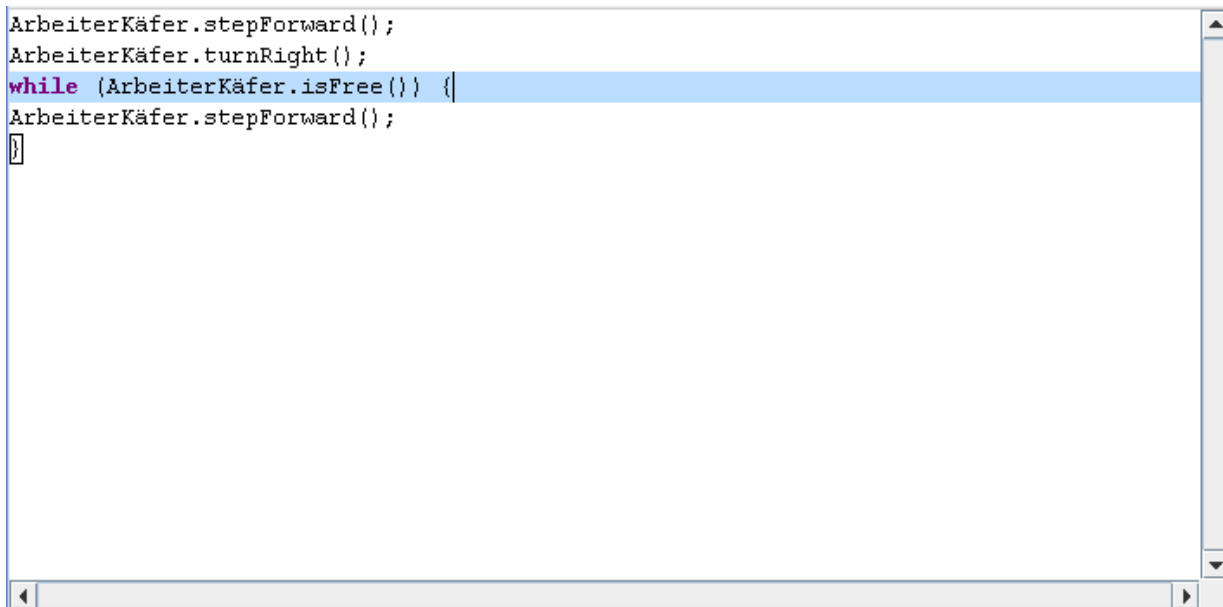
Er kann natürlich auch alles was ein normaler Käfer so kann. Er hat aber die spezielle Aufgabe bis zu fünf Kisten auf seinem Rücken zu tragen, die er von seinem Kollegen Arbeiterkäfer aufgeschnallt bekommt. Zum Ablegen der Kisten benötigt er jedoch keine Hilfe.



Scarabeus sensoricus - Fühlerkäfer

Der Spürhund unter den Käfern. Es gibt gewisse Kisten die mit einem Fragezeichen versehen sind. An diese traut sich kein anderer Käfer heran, da sie gefährliche Insektizide enthalten können. Der Arbeiterkäfer nimmt diese erst an sich, wenn dieser Käfer hier den Inhalt genauestens überprüft hat.

Der Code Editor



```
ArbeiterKäfer.stepForward();
ArbeiterKäfer.turnRight();
while (ArbeiterKäfer.isFree()) {
ArbeiterKäfer.stepForward();
}
```

Der *Code-Editor* ist die zentrale Anlaufstelle des Programms. Hier wird der Programmcode einfach von Hand eingegeben, oder besser: zusammengeklickt !

Wie man sich seinen Code zusammenklickt, erfährt man im übernächsten Absatz

Anmerkung : Die Sprache, die Bugs versteht, ist eine leicht abgespeckte Java-Version. Im Prinzip unterstützt Bugs im *Code-Editor* alle Elemente der Sprache Java bis auf eigene Klassen oder Interfaces. Es ist ohne weiteres möglich eigene Typen global oder lokal zu deklarieren, eigene Methoden zu schreiben oder Klassen zu importieren.

Der *Code-Editor* besitzt ein sog. „Syntax Highlighting“, welches verschiedene Java-Schlüsselworte aus Übersichtsgründen farblich hervorhebt.

Die Ausgabe-Konsole

Hier werden die Programm-Zustände oder Fehler ausgegeben.

Die Befehlsleiste

Die *Befehlsleiste* beinhaltet alle Befehle die man zum Zusammenklicken eines typischen Bugs-Programms benötigt. Sobald man in der *Käferauswahl* einen anderen Käfer auswählt, werden immer die Grundbefehle und die Sonderbefehle eines einzelnen Käfers angezeigt. Klickt man auf einen dieser Befehle wird der dazugehörige Code in den *Code-Editor* an der aktuellen Cursorposition eingefügt.

Folgende „anklickbare“ Befehle gibt es :

Die Grundbefehle

if

Dieser Befehl gehört weniger zu einem speziellen Käfer, sondern ist Bestandteil der Bugs-Sprache bzw. Java.

Entsprechung im Code : `if() {}`

for

Entsprechung im Code : `for(;;) {}`

while

Entsprechung im Code : `while() {}`



Veranlasst den Käfer einen Schritt nach vorne zu gehen, vorausgesetzt, es ist kein Hindernis im Weg. Ist ein Hindernis im Weg, endet die Ausführung des Bugs-Programms und es wird eine Fehlermeldung ausgegeben.

Dieser Befehl gehört zu den Grundbefehlen eines jeden Käfers.

Entsprechung im Code : `<Käfername>.stepForward();`



Veranlasst den Käfer sich um 90° nach rechts zu drehen.

Dieser Befehl gehört auch zu den Grundbefehlen eines jeden Käfers.

Entsprechung im Code : `<Käfername>.turnRight();`



Lässt den Käfer überprüfen, ob eine Mauer oder ein anderes Hindernis im Weg ist. Dieser Befehl ist ein Befehl mit Rückgabewert. Er liefert einen Wahrheitswert zurück. D.h.: Dieser Befehl kann zum Beispiel mit `if` oder `while` abgefragt werden.

Dieser Befehl gehört zu den Grundbefehlen eines jeden Käfers.

Entsprechung im Code : `<Käfername>.isFree();`

Spezielle Befehle der einzelnen Käfer

Arbeiterkäfer



Dieser Befehl nimmt einen Gegenstand auf (Kiste).

Entsprechung im Code : `<Käfername>.takeItem();`



Dieser Befehl stellt einen Gegenstand ab.
Der Transportkäfer hat diese Fähigkeit ebenfalls.

Entsprechung im Code : `<Käfername>.putItem();`

Transportkäfer

siehe Arbeiterkäfer

Fühlerkäfer



Dieser Befehl lässt den Käfer eine Kiste mit Fragezeichen auf Insektizide überprüfen.

Entsprechung im Code : `<Käfername>.checkItem();`

Was ist eine Umgebung ?

Eine *Umgebung* ist der Ort, an dem die Käfer leben. Sie kann folgende Elemente enthalten :



Rasen

Hierüber kann sich jeder Käfer frei bewegen.



Mauer

Eine Mauer stellt ein Hindernis dar.



Kiste

Eine Kiste stellt ebenfalls ein Hindernis dar, kann aber vom Transportkäfer versetzt werden



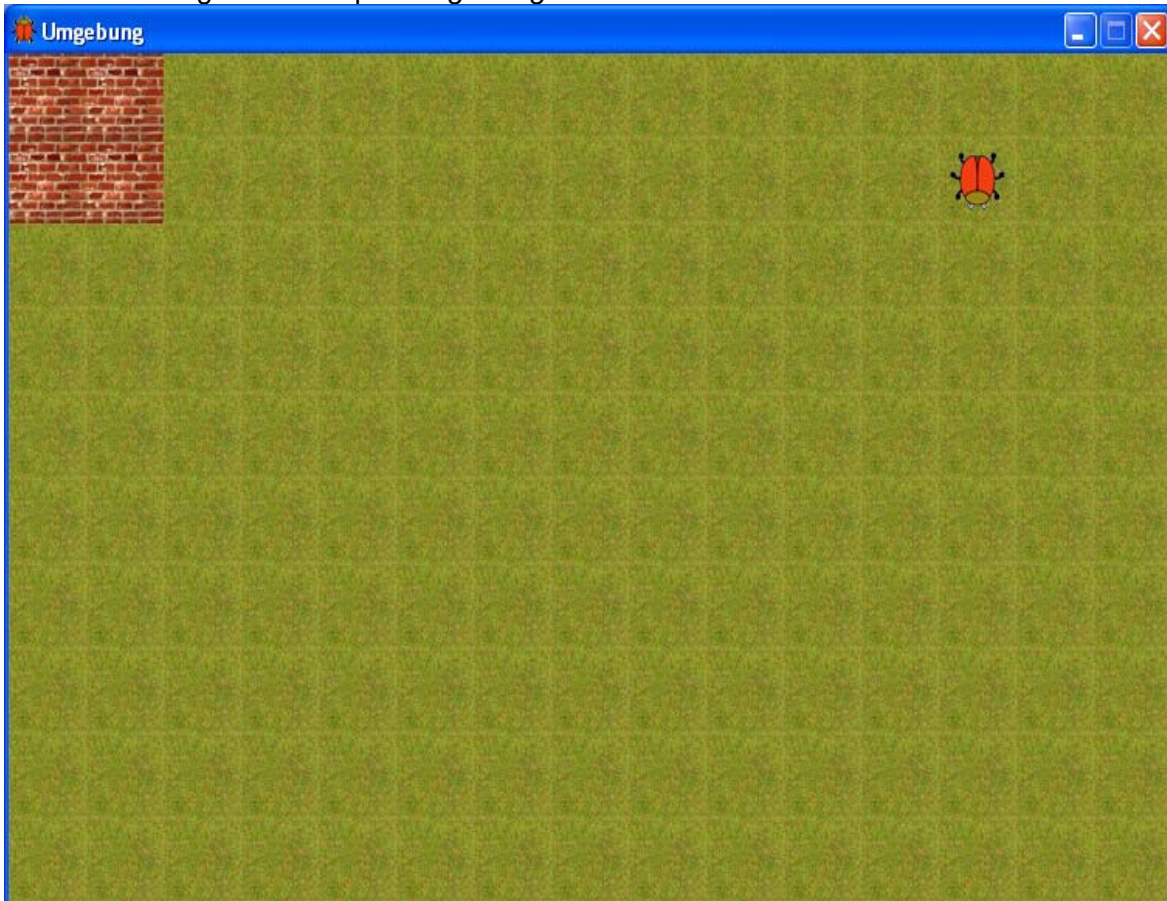
Kiste mit unbekanntem Inhalt

Diese Kiste kann einen gefährlichen Inhalt haben, deswegen weigert sich der Transportkäfer sie zu transportieren. Solch eine Kiste muss erst mit einem Fühlerkäfer überprüft werden. Ist ihr Inhalt ungefährlich verschwindet das Fragezeichen, andernfalls ziert sie ein rotes Ausrufezeichen.

Wie erstellt man eine Umgebung ?

Umgebungen kann man, wie die Überschrift vermuten lässt, auch selber erstellen. Die Eigenschaften einer Umgebung sind in XML definiert, man muss jedoch kein Experte sein, um sich eine eigene *Umgebung* zu erstellen.

Dieses Bild zeigt einen simple *Umgebung* :



Hier der Inhalt der dazugehörigen XML Datei :

```
<scenario name="simpleUmgebung">
  <staticitems>
    <item type="Mauer" x="0" y="0" />
    <item type="Mauer" x="0" y="1" />
    <item type="Mauer" x="1" y="0" />
    <item type="Mauer" x="1" y="1" />
  </staticitems>
  <variableitems>
    <item type="Robot" x="12" y="1" name="Käfer" facing="3" tool="" />
  </variableitems>
</scenario>
```

Man erkennt, das es nicht zu aufwändig ist, eine *Umgebung* zu erstellen.
Zur Erklärung :

Eine korrekte Umgebungsdatei besteht zunächst mal aus einem allumfassenden Block :

```
<szenario name="simpleUmgebung">
</szenario>
```

Dies wäre bereits eine *Umgebung*, jedoch vollkommen sinnlos, da es nichts enthält. Sehen würde man eine grüne Fläche und keinen einzigen Käfer.
Also fügt man in den Block einen neuen Block ein :

```
<szenario name="simpleUmgebung">
<staticitems>
</staticitems>
</szenario>
```

In diesen neuen Block fügt man wiederum Elemente ein, wie zum Beispiel Mauern.

```
<szenario name="simpleUmgebung">
<staticitems>
<item type="Mauer" x="0" y="0" />
</staticitems>
</szenario>
```

Mauern entsprechen dem Element item. Durch die Angabe von type="Mauer", drückt man aus, dass man eben diese Mauer darstellen will. Durch die Angabe von Koordinaten, beschreibt man die Position.

Das Koordinaten Paar wie im Beispiel (0/0) entspricht der linken, oberen Ecke der *Umgebung*. Eine *Umgebung* ist ein Rechteck von 15 Elementen in der Breite und 10 Elementen in der Höhe. Man beginnt mit dem Zählen jeweils bei 0.

Möchten man jetzt noch ein paar Käfer in die *Umgebung* setzen, ist das genauso einfach :

Man fügt einen neuen Block namens „variableitems“ ein :

```
<szenario name="simpleUmgebung">
<staticitems>
<item type="Mauer" x="0" y="0" />
</staticitems>
<variableitems>
</variableitems>
</szenario>
```

Dann fügt man den Käfer in diesen Block ein :

```
<szenario name="simpleUmgebung">
<staticitems>
<item type="Mauer" x="0" y="0" />
</staticitems>
<variableitems>
<item type="Robot" x="12" y="1" name="Käfer" facing="3" tool=""/>
</variableitems>
</szenario>
```

Zur Erklärung :

Ein Käfer ist nun auch wieder ein „item“ diesmal vom Typ „Robot“. Auch er erhält einen Namen und Koordinaten, jedoch die Attribute „facing“ und „tool“ sind neu.

Das Attribut „facing“ entspricht der Richtung in die der Käfer schauen soll. Hierfür sind bestimmte Zahlen definiert :

Rechts = 0
Unten = 1
Links = 2
Oben = 3

Diese Zählweise entsteht dann, wenn man von Rechts ausgehend im Uhrzeigersinn zählt.

Dann ist da noch das Attribut tool. Über dieses Element beschreibt man die Art des Käfers. Die Angabe von tool ist zwingend, wenn man also wie in diesem Beispiel einen einfachen Käfer erzeugen will, muss man tool= „“ angeben.

Werte für andere Käfer sind :

Arbeitskäfer = „Zange“
Transportkäfer = „Kiste“
Fühlerkäfer = „Sensor“

Möchte man mehrere Käfer oder Mauern etc. erstellen, fügt man diese einfach in ihre entsprechenden Blöcke ein. Unterschieden werden die einzelnen Käfer nicht durch ihre Art, sondern durch ihre Namen. Es ist also durchaus möglich mehrere Käfer des gleichen Typs in einer Umgebung zu haben.

Beispiel :

```
<scenario name="simpleUmgebung">
<staticitems>
  <item type="Mauer" x="0" y="0" />
  <item type="Mauer" x="1" y="2" />
</staticitems>
<variableitems>
  <item type="Robot" x="12" y="1" name="Käfer" facing="3" tool="" />
  <item type="Robot" x="12" y="1" name="Karl" facing="3" tool="Zange" />
</variableitems>
</scenario>
```

Letztendlich gibt es noch die Kisten, die von dem Arbeiterkäfer und dem Transportkäfer getragen sowie, im Falle einer Kiste mit Fragezeichen vom Fühlerkäfer überprüft werden können. Kisten werden genau wie die Käfer in den Block „variableitems“ eingetragen.

Beispiel :

```
<scenario name="simpleUmgebung">
<staticitems>
  <item type="Mauer" x="0" y="0" />
  <item type="Mauer" x="1" y="2" />
</staticitems>
<variableitems>
  <item type="Robot" x="12" y="1" name="Käfer" facing="3" tool=""/>
  <item type="Robot" x="12" y="2" name="Hans" facing="3" tool="Zange"/>
  <item type="Kiste" x="2" y="4" check="true" bad="true"/>
</variableitems>
</scenario>
```

Kisten besitzen nicht wie Käfer Namen und Werkzeuge, dafür aber wie oben gezeigt, die beiden Attribute „check“ und „bad“.

check = „true“ bedeutet, das die Kiste ein Fragezeichen erhalten soll, also überprüft werden soll.
check = „false“ wäre das Gegenteil.

bad = „true“ bedeutet, dass der Inhalt der Kiste hochgiftig für Käfer ist.

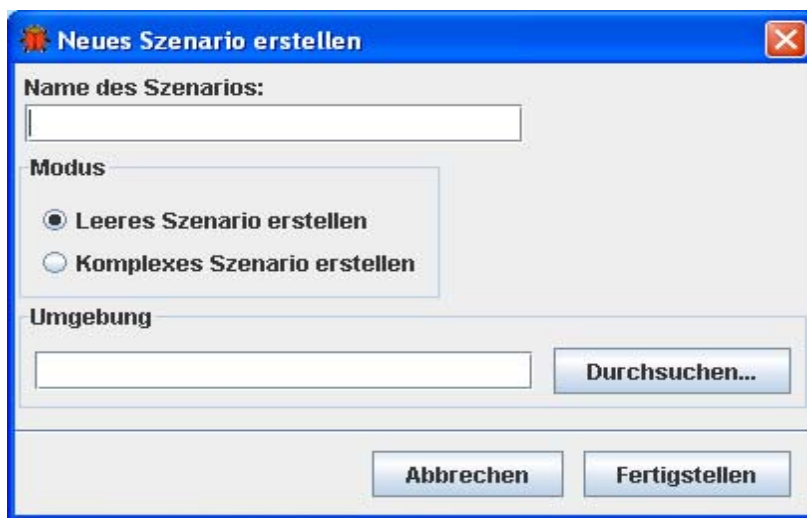
Was ist ein Szenario ?

Ein *Szenario* besteht aus dem Code, den man sich z.B. zusammengedrückt hat und aus einer *Umgebung*.

Dieser Ansatz resultiert aus der Vorstellung dieses Programm auch sinnvoll im Unterricht einsetzen zu können. Ein Lehrer könnte zum Beispiel eine *Umgebung* mit einer bestimmten Problemstellung entwerfen und lässt diese den Schüler programmatisch lösen.

Wie erstellt man ein Szenario ?

Um ein neues *Szenario* zu erstellen, klickt man im Dateimenü auf „Neues Szenario“. Es öffnet sich folgender Dialog :



Im Feld „Name des Szenarios“ trägt man den gewünschten Namen ein. Zusätzlich muss man eine vorhandene *Umgebung* wählen, entweder durch die Angabe eines Dateipfades oder durch einen Klick auf „Durchsuchen“.

Wählt man als Modus „komplexes Szenario erstellen“ wählt, werden zusätzlich im Code-Fenster zwei Leerfunktionen eingefügt. Diese benötigt man, wenn man eigene Methoden (Funktionen) im Code erzeugen will :

```
public void init() {  
  
}  
  
public void run() {  
  
}
```

Von diesen beiden Methoden muss nur eine mit Code gefüllt werden. Die sogenannte „Run()-Methode“. Man benötigt wie gesagt nur, wenn man eigene Methoden schreiben will. Der Grund dafür liegt darin, dass Bugs erkennen muss welche Methode das Hauptprogramm enthält. Bei Bugs ist das die „Run()-Methode“.

Die „Init()-Methode“ dient nur dazu, Dinge vor dem Programmstart zu initialisieren. Im Normalfall muss sie nicht mit Code gefüllt werden.