

# Squeak 3

## Grafisches Programmieren mit eToys

Heiko Schröder

1. März 2006



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
<b>2</b>	<b>Bearbeiten eines Morphs über Handles</b>	<b>4</b>
2.1	Namen ändern . . . . .	5
2.2	Kopieren . . . . .	6
2.3	Farbe ändern . . . . .	7
2.4	Rotieren . . . . .	8
2.5	Vergrößern . . . . .	9
2.6	Verschieben und Hochheben . . . . .	10
2.7	Menüs . . . . .	11
2.8	Kachel erzeugen . . . . .	12
<b>3</b>	<b>Steuern eines Morphs über den Viewer</b>	<b>13</b>
3.1	Starten des Viewers . . . . .	14
3.2	Kategorien des Viewers . . . . .	14
3.3	Schrittweise Aktionen . . . . .	15
3.4	Erzeugen eines Skriptes . . . . .	15
3.5	Starten und Stoppen eines Skriptes . . . . .	16
3.6	Zuweisungen . . . . .	17
3.7	Variablen . . . . .	18
3.8	Test . . . . .	19
3.9	Mehrere Skripte und Skriptsteuerung: Ein erstes Projekt . . . . .	20
3.9.1	Eine Art »Analyse« . . . . .	20
3.9.2	Eine Art »Design« . . . . .	20
3.9.3	Umsetzen in ein Programm . . . . .	20
3.9.4	Starten des Ablaufs . . . . .	22
3.9.5	Steuern angehaltener Skripte . . . . .	22
3.10	Turtle-Grafik mit eToys . . . . .	22
3.11	Zeichnen eines eigenen Morphs . . . . .	26
<b>4</b>	<b>Steuern mehrerer Morphe: Insekten am Seerosenteich</b>	<b>27</b>
4.1	Steuern eines Morphs durch einen anderen . . . . .	27
4.2	Abgleichen der Farbe . . . . .	29
4.3	Zulaufen auf einen bestimmten Punkt . . . . .	30
4.4	Erscheinen und Verstecken eines Morphs . . . . .	31
4.5	Ein Morph trägt das Kostüm eines anderen . . . . .	32
4.6	Siblings . . . . .	34

<b>5</b>	<b>Unterschiede zwischen eToys (<i>Self</i>) und Smalltalk*</b>	<b>34</b>
<b>6</b>	<b>Kleine Kontrollfragen</b>	<b>36</b>

## Abbildungsverzeichnis

1	eToys: Ändern des Namens . . . . .	5
2	eToys: Kopieren eines Morphs . . . . .	6
3	Ändern der Farbe . . . . .	7
4	Rotation mit dem blauen Handle . . . . .	8
5	Rotationszentrum eines Morphs . . . . .	8
6	Vergrößern mit dem gelben Handle . . . . .	9
7	Verschieben mit dem braunen Handle . . . . .	10
8	Hochheben mit dem schwarzen Handle . . . . .	10
9	Aufruf des Menüs mit dem roten Handle . . . . .	11
10	Kachel erzeugen mit dem orangefarbenen Morph . . . . .	12
11	Der Viewer eines Morphs . . . . .	13
12	Starten des Viewers mit dem türkisfarbenen Handle . . . . .	14
13	Das erste Skript . . . . .	16
14	Die Kategorie <i>Skripte</i> . . . . .	16
15	Einsetzen einer Kachel in ein Skript . . . . .	18
16	Die Test-Kachel . . . . .	19
17	Demonstration einer Verzweigung . . . . .	19
18	Das Skript <i>starten</i> . . . . .	21
19	Das Skript <i>bewegen</i> . . . . .	21
20	Das Skript <i>testeGeschw</i> . . . . .	21
21	Das Skript <i>testeY</i> . . . . .	21
22	Skriptkontrolle . . . . .	22
23	Die ausführliche Darstellung der Skriptsteuerung . . . . .	23
24	Eine Rosette mit eToys . . . . .	24
25	Zeichnen eines n-Ecks . . . . .	24
26	Zeichnen einer Rosette mit eToys . . . . .	25
27	Startskript für die Rosette . . . . .	25

# 1 Einleitung



Unsere nächste Reise führt uns zum ersten Male in die Welt der Turtle-Grafik. Du erinnerst Dich. Die erste Turtle von Seymour PAPERT war ein Roboter in Form einer Schildkröte. Diese Schildkröte konnte durch Nachrichten wie *forward*, *right* und *left* in ihrer Bewegung gesteuert werden. Zusätzlich gab es Nachrichten wie *up* und *down*. Mit diesen Nachrichten konnte der Stift der Turtle angehoben oder abgesenkt werden. Mit anderen Worten: Die Turtle konnte sich nur bewegen oder auch bei ihrer Bewegung zeichnen. Später wurde die Turtle natürlich nur noch durch eine Grafik am Bildschirm simuliert.

eToys<sup>1</sup> ist nun eine sehr starke Erweiterung dieser Turtle-Grafik. Eine Turtle kann ein beliebiger Morph sein. Ein Morph ist eine Grafik, die »lebt«. Ihre Eigenschaften können über einen Halo gesteuert werden. Sind Dir diese Dinge nicht mehr geläufig? Dann schaue bitte unbedingt noch einmal in dem Kapitel ?? auf Seite ?? nach. Die stärkste Erfindung von eToys ist aber das Skripting. Ursprünglich wird für das Steuern einer Turtle die Sprache *Logo* benutzt. Eine Sprache verlangt aber immer die Beherrschung einer gewissen Syntax (Rechtschreibung). Für die Jüngsten unter uns ist das unnötig schwierig. Stattdessen werden Programme durch das Ziehen von »Kacheln« zu so genannten *Skripten* zusammengebaut. Die Kacheln tragen schon von Beginn an eine richtige Syntax. Kleine Squeaker können sich so ganz auf das richtige Denken konzentrieren. Die Möglichkeiten von eToys sind ungeheuer groß. Fertige Projekte findest Du vor allem bei [www.squeakland.org](http://www.squeakland.org). Das Maskottchen von Squeakland ist denn auch die eToys-Maus, die dieses Kapitel eingeleitet hat. Squeakland liefert auch ein eigenes Squeak-Image aus, das auf eToys optimiert ist.

Was ist eigentlich der Grund für die Wahl von Mäusen als Maskottchen? Nun, das hat etwas mit Mickey Mouse zu tun. Morphic wurde von den Walt Disney Productions in Squeak eingebaut. Mit eToys kann nicht *alles* erreicht werden, was mit »echter« Smalltalk-Programmierung möglich ist. Dennoch ist eToys ein »echter Knaller«. Es gibt wohl keinen schöneren Einstieg in Squeak und in die Informatik überhaupt<sup>2</sup>.

## 2 Bearbeiten eines Morphs über Handles

Öffne zuerst ein Projekt. Auf diese Weise verhinderst Du ein »Vollmüllen« des Desktops. Außerdem kannst Du bei gutem Gelingen nur auf diese Weise Deine Arbeit sichern. Falls Du mit der Bezeichnung *Projekt* nichts mehr anfangen kannst, schaue bitte wieder im Kapitel ?? auf Seite ?? nach. Ja, und welchen Namen das Projekt tragen soll, bestimme selbst.

Es gibt zwei Arten von Morphen: selbst gezeichnete und vorgefertigte. Wir wollen zunächst uns nur mit vorgefertigten befassen. Du findest diese Morphe zum Beispiel in der roten Lasche mit der Aufschrift *Lager*. Ziehe dort bitte eine Ellipse heraus. Öffne dann mit der blauen Maustaste den Halo.

---

<sup>1</sup> Die Abkürzung bedeutet »educational toys«.

<sup>2</sup> In den USA gibt es auch außerordentlich gute Erfahrungen bei dem Einsatz von eToys in Grundschulklassen.

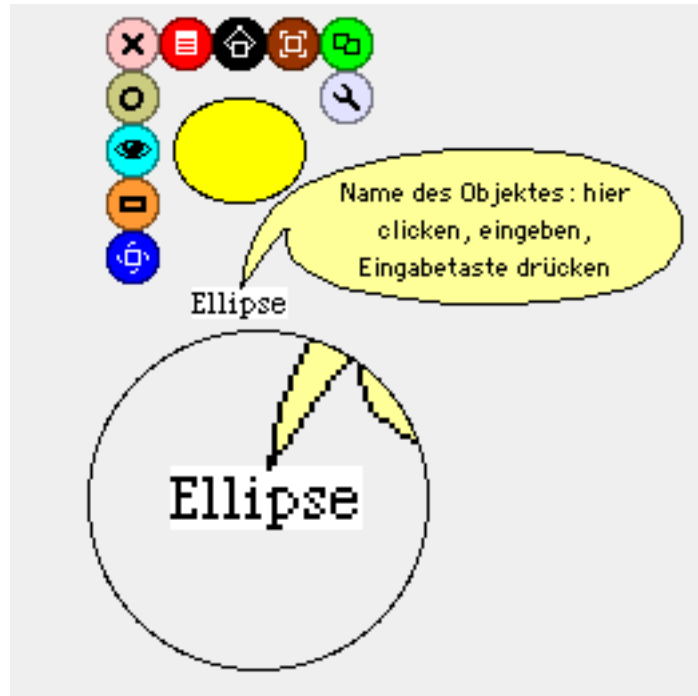


Abbildung 1: eToys: Ändern des Namens

## 2.1 Namen ändern

Ein neuer Morph trägt zunächst einen nicht sehr vielsagenden Namen. Diesen solltest Du in der Regel ändern. Einen Namen änderst Du auf die gleiche Art und Weise wie den Namen eines Projektes. Nennen wir unsere Ellipse mal wieder zur Abwechslung *Sveta*. Ja, Du darfst bei Morphen einen großen Namen verwenden.

Falls Du Dich noch nicht mit Smalltalk-Programmierung befaßt hast, überlies bitte diesen Absatz. Falls ja, stellst Du Dir sicherlich die Frage, ob *Ellipse* vielleicht eine Klasse ist. Nein, das ist nicht der Fall. Eine Ellipse ist bereits ein Objekt. Dieses Objekt stellt einen so genannten *Prototyp* dar. Was das genau bedeutet, sehen wir gleich.

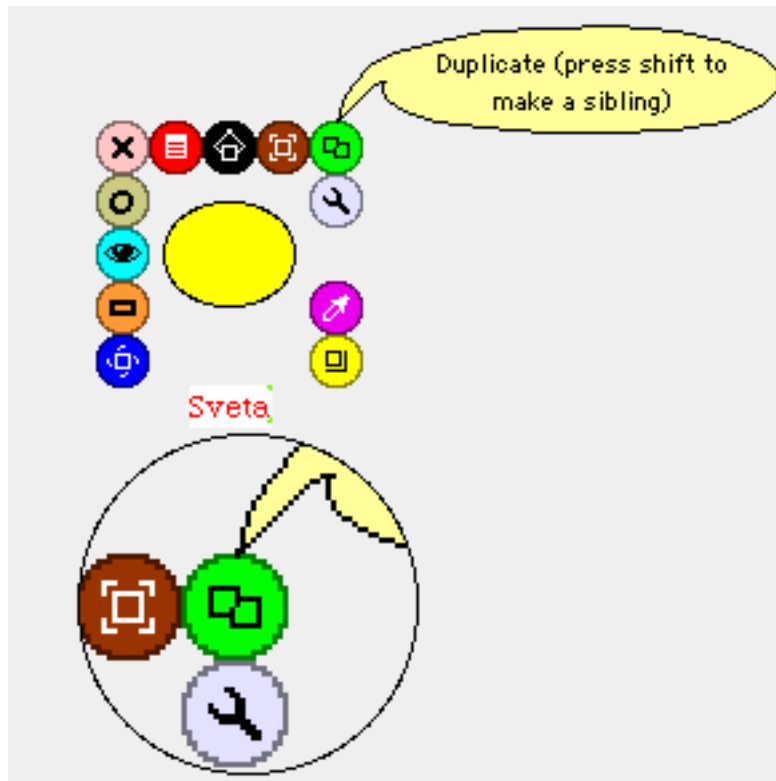


Abbildung 2: eToys: Kopieren eines Morphs

## 2.2 Kopieren

Mit dem grünen Handle kopierst Du einen Morph. Aber es scheint dafür zwei Möglichkeiten zu geben. Richtig.

**Duplikat:** Ein Duplikat ist eine »Verdopplung« des Morphs. Der neue Morph ist von dem alten unabhängig. Er ist eine einfache Kopie. Um ihn zu erhalten musst Du nicht nur das grüne Handle anklicken, sondern gleichzeitig die SHIFT-Taste drücken. Es wird an dieser Stelle noch nicht deutlich werden, wie das zu verstehen ist.

**Zwilling:** Ein Zwilling (engl. *sibling*) dagegen ist ein *abhängiger* Morph. Einen Zwillingsmorph erhältst Du ebenfalls über das grüne Handle. Aber *ohne* SHIFT-Taste.

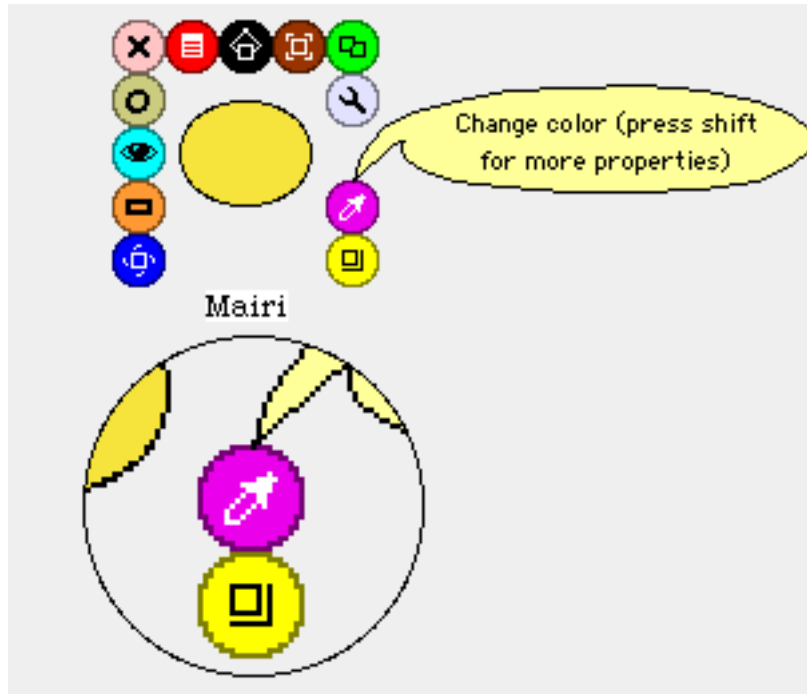


Abbildung 3: Ändern der Farbe

### 2.3 Farbe ändern

Die Farbe von vorgefertigten Morphs änderst Du über das rosafarbene Handle. Es öffnet sich dabei ein Auswahlfenster mit einem Farbverlauf und einer Pipette.



Mit dieser Pipette kannst Du aber nicht nur die Farbe aus dem Farbfenster wählen. Du kannst mit ihr auch die Farbe eines anderen Morphs holen. Dazu bewege Deine Pipette einfach nur auf den Morph und führe einen roten Mausklick aus. Die gewählte Farbe siehst Du in dem kleinen Quadrat oben rechts auf dem Rahmen. In dem Quadrat links daneben siehst Du den bisherigen Farbton.

Du kannst auf diese Weise allerdings nicht die Farbe eines selbst erstellten Morphs verändern.

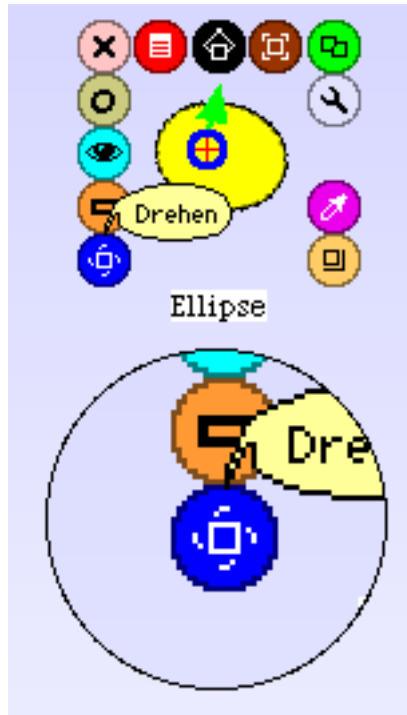


Abbildung 4: Rotation mit dem blauen Handle

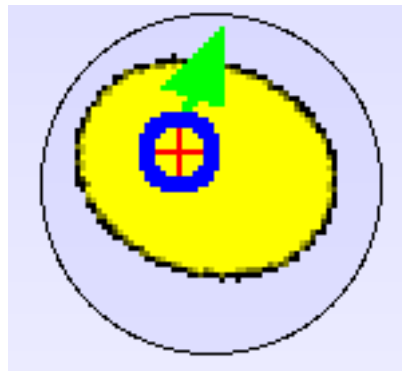


Abbildung 5: Rotationszentrum eines Morphs

## 2.4 Rotieren

Mit dem blauen Handle kannst Du einen Morph drehen. Dazu musst Du das Handle mit der roten Maustaste *ziehen*. Dabei zeigt sich in der Mitte des Morphs das Zentrum der Rotation. Es ist durch einen blauen Kreis mit einem roten Kreuz angedeutet. Das ist in der Abbildung 5 zu erkennen. Natürlich kannst Du die Lage dieses Zentrums verändern. Dazu musst Du *gleichzeitig* die SHIFT-Taste drücken und auf das Zentrum mit der roten Maustaste klicken und die Maustaste festhalten. Wenn Du jetzt die Maus verschiebst, wandert das Rotationszentrum mit.



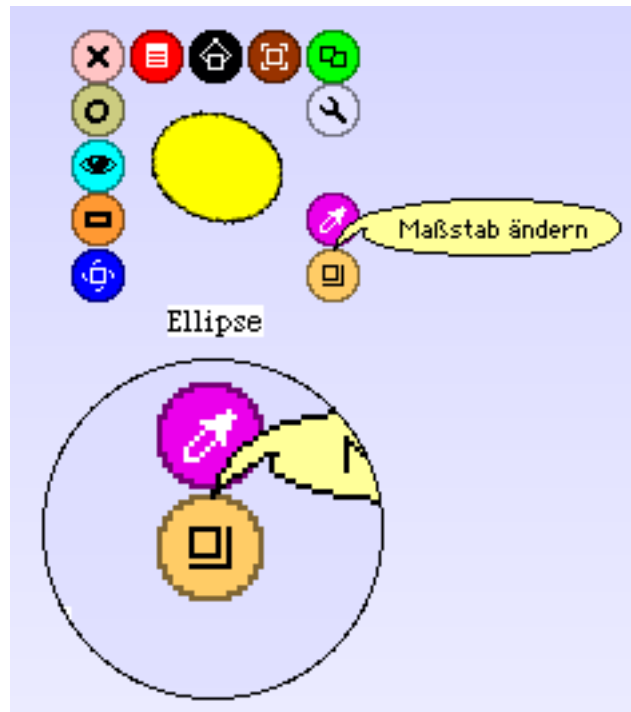


Abbildung 6: Vergrößern mit dem gelben Handle

## 2.5 Vergrößern

Mit dem gelben Handle vergrößerst Du. Hier gibt es keine Tricks wie etwa beim Rotieren oder beim Kopieren. Du hältst einfach die rote Maustaste auf dem gelben Handle fest und verschiebst die Maus. Der Morph verbleibt an der Stelle und verändert seine Größe.



Abbildung 7: Verschieben mit dem braunen Handle



Abbildung 8: Hochheben mit dem schwarzen Handle

## 2.6 Verschieben und Hochheben

Eine gewöhnliche Verschiebung wird mit dem braunen Handle vorgenommen. Nicht ganz klar ist wahrscheinlich der Unterschied zu dem schwarzen Handle. Bei einem einzelnen Morph ist er auch nicht vorhanden. Bei mehreren Morphhen kommt es jedoch zu Überdeckungen, wie in der Abbildung 8 . Der Halo gehört zu der unteren gelben Ellipse. Ein Morph, der an die Maus gebunden ist (aktiver) Morph überdeckt dabei immer den passiven. Das *Hochheben* unterscheidet sich von dem *Verschieben* nun dadurch, dass der verschobene Morph zugleich aktiviert wird. In der Abbildung war also zuletzt die blaue Ellipse aktiv. Danach wurde der Morph der gelben, halb verdeckten Ellipse aufgerufen.



Abbildung 9: Aufruf des Menüs mit dem roten Handle

## 2.7 Menüs

Mit dem roten Handle rufst Du das Kontextmenü des Morphs auf. Dieses erscheint – entgegen der üblichen Gepflogenheit - *nicht*, wenn Du mit der gelben Maustaste auf den Morph klickst. Allerdings erscheint nur das Kontextmenü des Morphs. Willst Du die ganze Welt untersuchen, in der sich der Morph befindet, so musst Du einen roten Mausklick bei gedrückter STRG-Taste in dem Projektfenster ausführen.

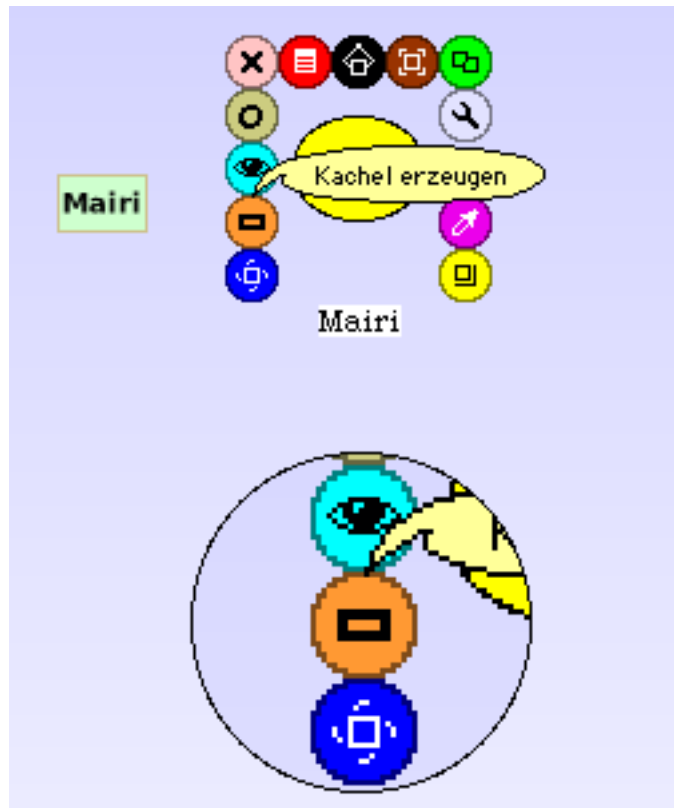


Abbildung 10: Kachel erzeugen mit dem orangefarbenen Morph

## 2.8 Kachel erzeugen

In dem folgenden Abschnitt werden wir die Frage nach der Programmierbarkeit des Verhaltens eines Morphs untersuchen. Vielleicht erinnerst Du Dich: Programme mit Hilfe so genannter Kacheln erstellt. Dabei *kann* es vorkommen, dass Du den Morph selbst als Kachel benötigst. Dies ist zum Beispiel dann der Fall, wenn Du zwei Morphe zusammenstoßen läßt. Wenn unser Morph *Mairi* heißt, so wird also der andere Morph untersucht, ob er mit dem Morph *Mairi* zusammengestoßen ist. In der Abbildung 10 siehst Du links neben dem orangefarbenen Button die Kachel. Diese Kachel kann nun in ein Skript gezogen werden. Was ein Skript ist, erfährst Du jetzt gleich.

An dieser Stelle möchtest Du die Kachel sicherlich wieder loswerden. Nun, eine Kachel ist ein Morph. Damit ist die Frage bereits beantwortet. Über den Halo kannst Du mit der rosafarbenen Kachel mit dem Kreuz X jeden Morph löschen.



Abbildung 11: Der Viewer eines Morphs

### 3 Steuern eines Morphs über den Viewer

Nun beginnen wir mit dem eigentlichen Programmieren. Bisher haben wir nur die *Eigenschaften* eines Morphs geändert. Sozusagen »von außen«. Wir haben Kopien erzeugt und wissen, wie wir ihn durch eine Kachel darstellen können. Jetzt lernst Du das Senden von *Nachrichten* an einen Morph.

Wir haben es schon mehrmals gesagt: Bei eToys werden Nachrichten durch eine *Kachel* dargestellt. In der Abbildung 11 siehst Du einen geöffneten Viewer unserer Ellipse »Mairi«. Du wirst gleich lernen wie Du ihn erhältst. Die grünen Streifen in diesem Viewer sind die Kacheln. Sie sind *beweglich* und können zu so genannten *Skripten* zusammengezogen werden. Diese Skripte sind dann die eToys-Programme.

Jede Kachel enthält bereits Anweisungen, die der Morph versteht. Es kann Dir also *nicht* passieren, dass Du eine nicht verständliche Nachricht sendest. Zum Beispiel eine Nachricht, die Rechtschreibfehler enthält, die also gegen die Regeln von Smalltalk verstößt. Du kannst Dich also völlig auf das richtige Denken konzentrieren. Du hast damit im Vergleich zu einem Smalltalk-Programmierer einen gewissen Vorteil. Ein kleiner Nachteil soll aber nicht verschwiegen werden: Mit eToys kannst Du *unglaublich* viel programmieren. Aber ein »Smalltalker« kann natürlich mehr erreichen. Daher wirst Du Dich sicherlich später mit dem »wirklichen« Smalltalk beschäftigen wollen.

Doch vergessen wir zunächst Smalltalk. Das gilt auch für solche Leser, die diese Sprache bereits kennen. Für diese Leser gibt es an dieser Stelle einen wichtigen Hinweis, damit sie sich nicht zu sehr wundern: eToys arbeitet – wie übrigens das ganze Morphic-System von Squeak – mit einem Smalltalk-Dialekt, der *Self* heißt. Bei *Self* ist einiges anders als

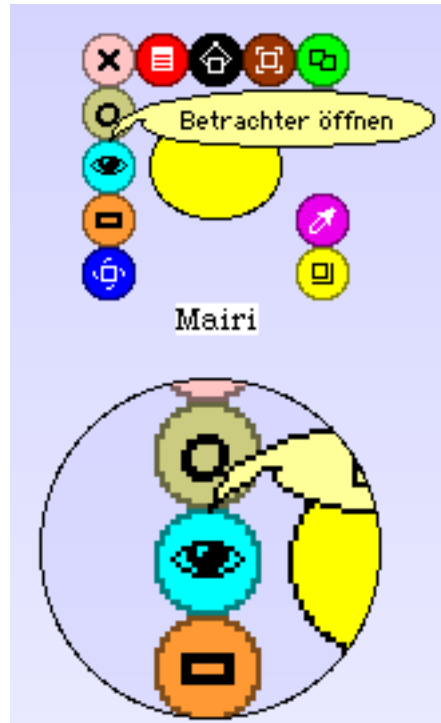


Abbildung 12: Starten des Viewers mit dem türkisfarbenen Handle

bei purem Smalltalk. Für Anfänger sind dies Unterschiede nicht wichtig. Wer aber schon Smalltalk-Erfahrungen hat, sollte dann und wann einen Blick auf den Abschnitt 5 auf Seite 34 werfen. Dort werden die Unterschiede erklärt.

### 3.1 Starten des Viewers


Den Viewer startest Du mit dem türkisfarbenen Handle. In unserem Beispiel ist der Morph eine Ellipse mit dem Namen »Mairi«. Der Viewer sucht sich automatisch eine Position am rechten Rand des Squeak-Desktops. Links am Viewer erkennst Du einen kleinen Reiter mit dem verkleinerten Bild des Morphs. Über diesen Reiter kannst Du den Viewer verschieben oder schließen.

Wenn Du den Viewer aus irgendwelchen Gründen wieder loswerden möchtest – ich wüßte allerdings nicht, aus *welchem* Grund – so stellst Du fest, dass das Ganze aus drei Morphs zusammengesetzt ist. Der eigentliche Viewer kann – wie jeder Morph – über den Halo gelöscht werden. Doch wie Du siehst: Der Reiter verschwindet nicht. Ebenso scheint der Viewer in einem grünlichen Container eingebettet zu sein. Sowohl den Container als auch der Reiter sind eigene Morphs. Sie müssen entsprechend separat gelöscht werden.






### 3.2 Kategorien des Viewers

Schauen wir uns ein wenig um. Wir setzen voraus, dass Du die Sprache auf »Deutsch« umgestellt hast. Wenn das nicht der Fall ist, schaue in den Grundlagen (Squeak 2) noch einmal genau nach, wie das geht. Wenn Du die Sprache umgestellt hast, sollten fast alle Kacheln Anweisungen in deutscher Sprache zeigen (z.B.: Mairi gehe vorwärts um. . .).

Alle verständlichen Nachrichten sind in Kategorien unterteilt. Sehr wahrscheinlich findest

Du nach dem Starten die Kategorie  **Einfach** an erster Stelle. Du kannst den grauen Knopf mit der Bezeichnung »Einfach« anklicken. Danach erscheint ein Menü. Aus diesem Menü kannst Du beliebige andere Kategorien auswählen. Durch Anklicken der grünen Pfeile springst Du zu einer der benachbarten Kategorien.

### 3.3 Schrittweise Aktionen

Und nun sendest Du die erste Nachricht an die Ellipse *Mairi*. Suche die Kachel mit der Aufschrift   **Mairi gehe vorwärts um**  **5** . Du findest sie in der Kategorie »Einfach«. Diese Kachel enthält am linken Rand das gelbe Feld  mit dem Ausrufezeichen. Wenn Du dieses Feld anklickst, sendest Du Mairi *einmal* die auf der Kachel angegebene Nachricht. Mairi sollte sich bei einem solchen Klick ein wenig nach oben bewegen.





Was passiert genau? Es ist sehr wichtig, dass wir so weit wie möglich *verstehen*, was abläuft. Die Kachel besteht tatsächlich aus *drei* einzelnen Kacheln:

1. Die erste trägt den Namen des empfangenden Objekts. Das ist also in diesem Falle *Mairi*.
2. Mairi kann nun auf sehr viele *verschiedene* Nachrichten reagieren. Aus dieser Unmenge an möglichen Reaktionen sortiert die zweite Kachel eine ganz bestimmte aus. Diese Kachel mit der Aufschrift »gehe vorwärts um« heißt deshalb auch *Nachrichtenselektor*.
3. Da wir nun eben einen Begriff benutzt haben, den auch Profis verwenden, gleich noch einer hinterher: Als dritte Kachel siehst Du eine *Zahl* als *Argument*. Dieses Argument ist für Mairi notwendig. Sie weiß natürlich, was es heißt, vorwärts zu gehen. Aber für die *exakte* Reaktion reicht dies noch nicht aus.

Nun, was die grünen Pfeile bedeuten, kannst Du Dir denken. Aber wozu ist der *rechte* Pfeil da? Ups. Klicken wir ihn an, haben wir die *gesamte* Kachel plötzlich »in der Hand«. Ziehe sie einfach auf den Desktop und lasse sie fallen. Es wird etwas passieren, das wir nun erläutern.

### 3.4 Erzeugen eines Skriptes

Du hast soeben Dein erstes Skript erzeugt. Noch einmal genauer. Bewege die Maus über unsere Kachel im Viewer, *ohne* eine Maustaste zu drücken. Dir wird auffallen, dass die

Kachel   **Mairi gehe vorwärts um**  **5**  jetzt einen roten Rand trägt. Wenn Du jetzt irgendwo innerhalb dieses Bereiches die Kachel anklickst, ziehst Du von ihr eine Kopie auf den Desktop. Das Ergebnis sollten dann so aussehen wie in der Abbildung 13 auf der nächsten Seite .

In der obersten Zeile stehen sehr viele verschiedene Symbole. Wir werden sie alle kennenlernen. Klicke jetzt noch einmal den rechten grünen Pfeil neben der 5 an.

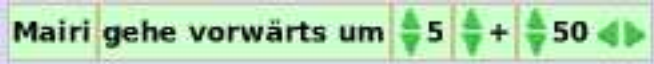
Aha.  Die Zeile verlängert sich und es ist möglich, eine Zahl zu addieren, zu subtrahieren und noch mehr. Wenn Du mit der roten



Abbildung 13: Das erste Skript

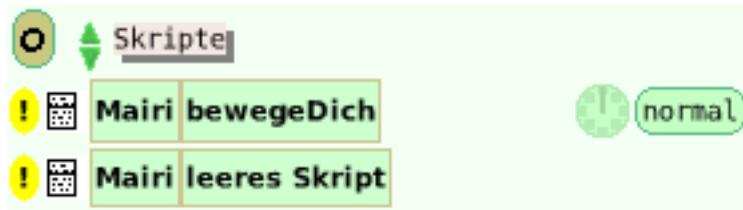


Abbildung 14: Die Kategorie *Skripte*


Maustaste lange auf das Pluszeichen klickst, so siehst Du in einem Menü *was* alles möglich ist. Sehr wahrscheinlich wird bei Dir hinter dem Additionszeichen eine 1 und keine 50 stehen. Du kannst den Wert nicht nur über die vertikalen grünen Pfeile ändern. Auf dieselbe Weise wie bei der Erstellung eines Projektes kannst Du die 1 überschreiben.

Durch Überschreiben änderst Du auch den nicht sehr vielsagenden Namen »Skript1« um. Zum Beispiel in »bewegeDich«. Wenn Du jetzt auf das Ausrufezeichen im gelben Feld klickst, macht Mairi schon einen recht ansehnlichen »Hopser«. Tatsächlich hast Du Mairi die Nachricht *Mairi bewegeDich.* gesendet. Diese Nachricht ist mit *Mairi gehe vorwärts um 5+50* gleichbedeutend. Aber nur deshalb, weil das Skript bisher nur diese eine Kachel enthält.

Der Sinn der Addition ist an dieser Stelle sicherlich nicht einsichtig. Wieso ändern wir die 5 nicht gleich auf 55? Natürlich. Bei unserem einfachen Beispiel wäre es das Gleiche. Wir werden jedoch bald sehen, dass wir die Kacheln mit den Zahlen durch andere Kacheln ersetzen können, die Namen für Zahlen tragen, deren Wert sich im Laufe der Zeit *verändert*.

Es gibt nun noch eine elegantere Art und Weise, ein Skript zu öffnen. Dies zeigt die Abbildung 14. In der Kategorie »Skripte« findest Du nicht nur sämtliche, bereits angelegten Skripte. Sondern es gibt dort auch die Kachel mit dem Selektor *leeres Skript*. Wenn diese Kachel herausgezogen wird, erhältst Du wirklich ein leeres Skript. Das kann notwendig sein, wenn Du Skripte erzeugen willst, die sich *gegenseitig* aufrufen.



Durch ein Klick auf das Symbol  verschwindet das Skript zwar von der Bildfläche. In der Kategorie *Skripte* ist es aber nach wie vor zu finden. Du öffnest das Skript wieder, indem Du seine Kachel zurück auf den Desktop ziehst.

### 3.5 Starten und Stoppen eines Skriptes

So, und nun lassen wir unser Skript *bewegeDich* laufen. Der Klick auf das Ausrufezeichen führt es ja nur ein einziges Mal aus. Das Starten eines Skriptes hat sicherlich etwas mit der Uhr zu tun. Natürlich. Wenn Du die Uhr anklickst, wird Mairi mehr oder weniger schnell über den Bildschirm laufen. Die Uhr kann sich in drei verschiedenen Zuständen befinden:



Der Zustand *normal* zeigt an, dass das Skript nicht läuft. Es muss eine *explizite* Nachricht durch einen Mausklick auf die Uhr oder eine Nachricht von einem anderen



Skript erhalten.




Ein laufendes Skript führt eine bestimmte Zahl von Ticks in der Sekunde aus. Über die Kachel *Mairi gehe vorwärts um 5+50* stellst Du genau genommen *nicht* die Geschwindigkeit ein. Sondern vielmehr die Schrittweite. Die Geschwindigkeit wird eher über die Zahl der Ticks festgelegt. Diese kannst Du ändern, indem Du die Uhr bei festgehaltener Maustaste anklickst. Es erscheint ein entsprechendes Kontextmenü. Ändere die Zahl der Ticks auf 100 ab und Sorge dafür, dass Mairi nur noch einen Schritt von 5 Bildschirmpunkten ausführt. Dazu lasse über die grünen Pfeiltasten die 50 wieder verschwinden. Mairi läuft wesentlich gleichmäßiger über den Bildschirm.



Du hältst ein laufendes Skript natürlich wieder durch einen Mausklick auf die Uhr an. Im Gegensatz zu dem Zustand *Normal*, ist ein angehaltenes Skript immer noch sozusagen »aktiv«. Angehaltene Skripte können *gleichzeitig* wieder gestartet werden, normale dagegen nur einzeln.

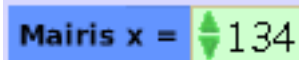
Hinter den farbigen Feldern, die den Zustand beschreiben, findest Du eine Möglichkeit, den Zustand des Skripts manuell zu ändern.

### 3.6 Zuweisungen

In dem Viewer siehst Du eine ganze Reihe von Kacheln, die einen grünen Pfeil auf violettem Hintergrund tragen . Bei diesen Kacheln handelt es sich genau genommen nicht um Nachrichten an das Objekt *Mairi*. Schauen wir uns zum Beispiel die Kachel mit der Aufschrift *Mairis x* einmal genauer an. Du findest sie in den Kategorien *Einfach* und *Geometrie*.

Schon allein die Bezeichnung *Mairis* statt *Mairi* zeigt Dir, dass diese Kacheln eine *Eigenschaft* beschreiben und keine Nachrichten darstellen. Etwas verwirrend scheint auf den ersten Blick zu sein, dass Du diese Kacheln auf zwei verschiedene Art und Weisen herausziehen kannst.

1. Ziehst Du *nur* den Teil *Mairis x* allein heraus, so sieht die Kachel auf dem Desktop



so aus: *außerhalb eines Skripts* nichts anzufangen ist. Du hast nichts anderes getan als eine *Eigenschaft* abzufragen. Die Eigenschaft der x-Position von Mairi. Bei Dir kann die angegebene Zahl natürlich anders sein. Je, nachdem, an welcher Position sich bei Dir Mairi befindet. Diese Information aber findest Du aber auch im Viewer selbst. Lasse die Kachel also wieder über den Halo verschwinden.

2. Du kannst aber auch die *ganze* Kachel herausziehen, indem Du *vor* dem Ziehen die Maus auf den besagten Pfeil bewegst. Es öffnet sich ein neues Skript und *innerhalb*



dieses Skripts erscheint die Kachel in der Form *innerhalb*. Hierbei handelt es sich um eine echte Anweisung. Sie besagt, dass der Eigenschaft *x* von *Mairi* ein Zahlenwert *zugewiesen* wird.

Die erste Methode bedeutet also die Abfrage eines gerade bestehenden *Zustands*. Die zweite Methode ist die Möglichkeit, diesen Zustand zu *verändern*.

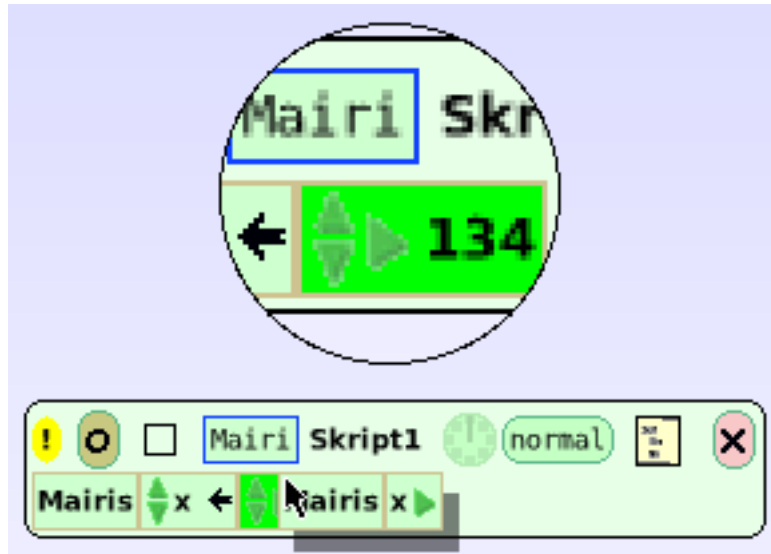


Abbildung 15: Einsetzen einer Kachel in ein Skript

Wir können nun beides kombinieren. Nachdem Du mit der zweiten Methode ein Skript geöffnet hast, ziehe nach der ersten Methode *Mairis x* über die Zahl (in meinem Beispiel 134). Sobald Du Dich mit der Kachel über der Zahl befindest, erhält diese einen grünen Hintergrund. Das siehst Du in der Abbildung 15. Solange Du diesen Hintergrund noch *nicht* siehst, lasse die Kachel nicht fallen. Eventuell musst Du sie noch ein wenig hin und her bewegen. Sobald Du der grüne Hintergrund erscheint, kannst Du die Kachel fallenlassen. Sie begibt sich mit einem deutlich hörbaren »Knall« an den richtigen Ort. Dieser »Kachel-Knall« ist die wichtige akustische Rückmeldung dafür, dass die Kachel richtig sitzt. Wenn Du jetzt mit dem rechten Pfeil wieder +50 addierst, so heißt das: »Erhöhe den aktuellen Zustand von Mairis x um 50«.

Schauen wir uns dazu ein interessanteres Beispiel an. Ziehe auf die beschriebene Art *Mairis y* in Dein Skript *bewegeDich*. Ziehe dann mit dem rechten grünen Pfeil die Erweiterung auf. Statt +50 schreiben wir /10. Das Zeichen / ist eine so genannte Ganzzahl-Division. Da es keine halben Schritte geben kann, ist eine »echte« Division // nicht richtig. Das Zeichen \\  
bedeutet eine Modulo-Division, die den *Rest* einer Division angibt.

Das richtige Skript sieht jetzt so aus:



Und was passiert? Mairi wird schneller und schneller. Warum, kannst Du Dir sicherlich erklären. Die 10 ist nur dazu da, dass die Zunahme der Geschwindigkeit nicht *zu* groß ist.

### 3.7 Variablen


*Mairis x* und *Mairis y* sind Namen für Objekte. Statt *Namen* wird auch *Variablen* gesagt. Neben der Variable *Mairi* selbst, stehen Dir alle Variablen zur Verfügung, die mit dem Wort *Mairis* beginnen. Das sind alle Kacheln mit dem beschriebenen Zuweisungspfeil.



Abbildung 16: Die Test-Kachel




Abbildung 17: Demonstration einer Verzweigung

Gibt es Möglichkeiten, *eigene* Variablen einzurichten? Ja. Dazu klicke in dem Viewer auf das v-Symbol  **Mairi** am oberen Rand des Viewers. Es öffnet sich ein Eingabefenster. Gib den Namen *geschw* ein und klicke dann auf OK. Nach einer kurzen Zeit öffnet sich eine neue Kategorie (!) mit der Bezeichnung *Variablen*. Dort erscheint die neue Variable *geschw*. Natürlich wird sie die Bedeutung einer Geschwindigkeit haben. Wir werden es gleich sehen.

### 3.8 Test

Neben dem ständigen Wiederholen eines gleichen Vorgangs, gibt es als zweites wichtiges Element die *Entscheidung* oder die *Verzweigung* in jeder Programmiersprache. Bei eToys heißen solche Entscheidungen *Test*.

Die Test-Kachel findest Du in jedem Skript unter dem Symbol . Auch diese Kachel muss gezogen werden. Sie sieht so aus wie in der Abbildung . Hinter dem Wort *Test* muss eine einzige Kachel eingefügt werden, die einen Wahrheitswert abfragt. Öffne ein leeres Skript und gib ihm den Namen *testDemo*. Ziehe dann wieder unsere Bewegungskachel *Mairi gehe vorwärts um 5* in das Skript hinein.

Ziehe dann die Test-Kachel in das Skript hinein. Achtung: Auch hier muss das »grüne Signal« erscheinen. Manchmal musst Du die Test-Kachel dazu ein wenig bewegen. Achte auf den »Kachelknall«.

Hinter das Wort *Test* richte eine Kachel ein, die abfragt, ob *Mairis y* größer als 400 ist. Hinter dem Wort *Ja* werden nun alle Nachrichten untergebracht, die ausgeführt werden sollen, falls die Abfrage den Wert *wahr* ergeben hat. *Mairi* soll in diesem Fall ein Signal erzeugen. Im Erzeugen von Geräuschen ist Squeak wohl kaum zu überbieten. Ziehe die Kachel *Mairi mache Geräusch* in die Testkachel hinein. Suche Dir ein Geräusch Deines Geschmacks aus.

Solange *Mairis* Wert aber noch nicht 400 erreicht hat, soll sie sich einfach nur nach oben bewegen. Das fertige Skript zeigt die Abbildung 17 . Bevor Du das Skript ausführst, schließe aber alle Fenster, damit Deine Nachbarn nicht von dem Krach gestört werden.

### 3.9 Mehrere Skripte und Skriptsteuerung: Ein erstes Projekt

Eine ganz wesentliche Kunst besteht darin, die verschiedenen Aufgaben auf *verschiedene* Skripte zu verteilen. Vor allem werden wir nun immer mehr dazu übergehen, uns *vorher* genau zu überlegen, was wir eigentlich wollen.

#### 3.9.1 Eine Art »Analyse«

1. Unser erstes »richtiges« Programm soll *Mairi* nach oben laufen lassen.
2. Dabei soll sich die Geschwindigkeit um einen gleibenden Betrag erhöhen. Wenn die Geschwindigkeit den Wert 100 übersteigt, soll ein akustisches Signal ertönen. Die Geschwindigkeit wird daraufhin wieder auf ihren Anfangswert zurückgesetzt.
3. Ferner darf *Mairis* y-Wert die Zahl 600 nicht überschreiten. Wenn dies der Fall ist, so wird dieser Wert auf den Anfangswert zurückgesetzt.

Wenigstens in *dieser* Form sollte *vor* dem ersten Skripting das eigentliche Vorhaben dokumentiert werden. In Squeak 5 erlernst Du die ganze Kunst dieser Vorüberlegungen.

#### 3.9.2 Eine Art »Design«

Aus unserem Plan ergibt sich die Notwendigkeit von vier Skripten. Was sollen die einzelnen Skripte können?

**starten** soll die Anfangswerte festlegen. Dies sind *Mairis x*, *Mairis y*, *Mairis geschw* und *Mairis beschl*. Die letzten beiden Variablen *Mairis geschw* und *Mairis beschl* sind vorher einzurichten. *beschl* steht natürlich für die *Beschleunigung*. Als Werte geben wir vor: *Mairis x=200*, *Mairis y=100*, *Mairis geschw=5* und *Mairis beschl=2*. Das Skript *starten* startet am Ende des Ablaufs das Skript *bewegen*.

**bewegen** Die wesentliche Aufgabe dieses Skripts ist die Bewegung von *Mairi* um einen Schritt mit der gerade eingestellten Geschwindigkeit. Vorher werden die Skripte *testeY* und *testeGeschw* jeweils *einmal* aufgerufen. Ganz zu Beginn muss *bewegen* aber dafür sorgen, dass das Skript *starten* angehalten wird. Sonst setzt *starten* jedes Mal die Eigenschaften auf die Anfangswerte zurück.

**testeGeschwindigkeit** erzeugt ein akustisches Signal und setzt die Geschwindigkeit auf den Anfangswert zurück, falls der Wert 100 überschritten wird.

**testeY** setzt *Mairis y* zurück, falls der Wert 600 überschritten wird.

#### 3.9.3 Umsetzen in ein Programm

Schreiben wir nun die vier Skripte. Zuerst müssen wir die beiden Variablen *geschw* und *beschl* einrichten, falls das noch nicht geschehen ist. Danach erzeuge die vier Skripte zunächst als *leere Skripte*. Neu ist, dass sich einige Skripte *gegenseitig* aufrufen. Die dafür notwendigen Nachrichten findest Du in der Kategorie *Skriptverwaltung*.

Versuche jetzt, die vier Skripte einmal selbst zusammenzustellen. Schaue erst dann auf die Abbildungen.

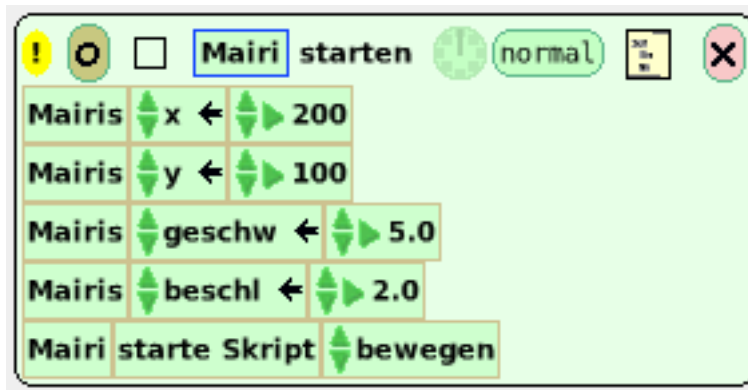


Abbildung 18: Das Skript *starten*



Abbildung 19: Das Skript *bewegen*

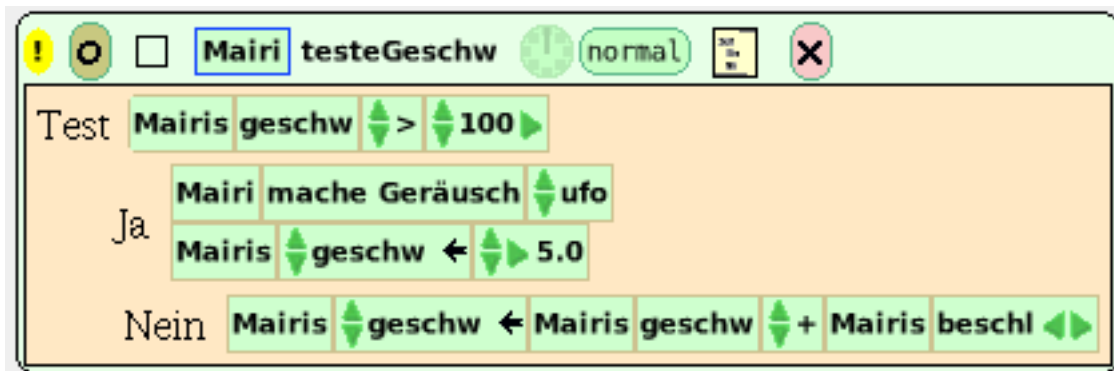


Abbildung 20: Das Skript *testeGeschw*

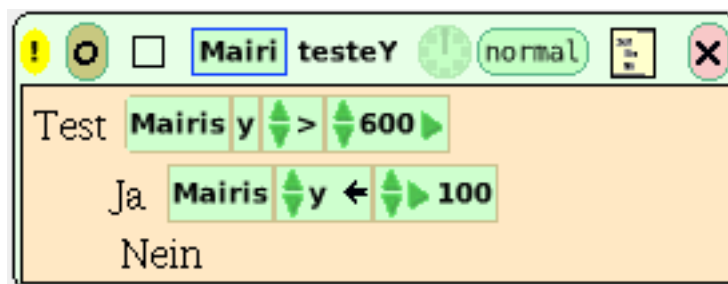


Abbildung 21: Das Skript *testeY*

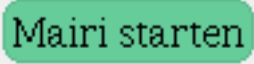


Abbildung 22: Skriptkontrolle

### 3.9.4 Starten des Ablaufs

Offenbar gibt das Skript *starten* den Startschuss. Natürlich kannst Du die Uhr anklicken. Aber es gibt eine sehr viel bessere Methode. Jedes Skript verfügt über ein Kontextmenü.

Dieses Menü erreichst Du über den eingerahmten Namen   **Mairi starten** in der oberen Leiste des Skriptes. Über das Kontextmenü kannst Du Dir einen Start-Button

 einrichten. Achte darauf, dass dieser Button nur über den Halo verschoben werden kann, wenn er einmal gesetzt wurde.

Starte bitte jetzt noch nicht, sondern lies Dir erst einmal durch, wie Du alles wieder anhalten kannst.

### 3.9.5 Steuern angehaltener Skripte

In der blauen Lasche mit der Aufschrift Geräte verbirgt sich ein wunderbares Tool: Die *Skriptkontrolle*. Die Abbildung 22 zeigt die Schaltfläche im Normalzustand. Über den blauen Kreis ganz rechts kannst Du eine ausführliche Ansicht anfordern. Tue dies jetzt. Noch wird nichts zu sehen sein, weil sozusagen kein Skript aktiv ist. Du kannst dies ändern, indem Du die Markierung für »nur laufende Skripte« entfernst.

Das aber ist nicht nötig. Befindet sich irgendein Skript im angehaltenen Zustand, so wird es schon auftauchen. Angehaltene Skripte sind *aktiv*. Nun ist alles gesagt. Über den Start-Button aktivierst Du das Skript *starten*. Dann unterhalten sich nur noch *bewegen* und die Testskripte. Wenn Du mit *stop* den Ablauf anhältst, kannst Du ihn mit *go* natürlich wieder aktivieren. Aber es beginnt alles von vorn. Da das sich Skript *starten* in angehaltenem Zustand befindet, wird es mit *go* ebenfalls wieder gestartet. Die Abbildung 23 auf der nächsten Seite zeigt Dir die ausführliche Darstellung der Skriptsteuerung während des Betriebs.

Falls irgendein Fehler vorliegen sollte, den Du nicht finden kannst, lade das Projekt *eToys-Mairi1.pr*. Dieses findet sich in demselben Paket wie die PDF-Datei von Squeak 3.

## 3.10 Turtle-Grafik mit eToys

Eigentlich sind alle Morphe Nachfahren der Turtle-Grafik, die Seymour PAPERT in den sechziger Jahren am MIT entwickelt hat. Das haben wir schon mehrfach in diesem Tutorial gesagt. Turtles sind Grafiken, die ihre Bewegungen aufzeichnen können.

Wir wollen uns mit einem sehr bekannten Problem beschäftigen: Unsere Ellipse soll ein beliebiges *n-Eck* zeichnen. Aber noch mehr: Sie soll dieses *n-Eck* nicht nur *einmal* zeichnen. Wir geben eine beliebige Anzahl vor. Nach jeder beendeten Zeichnung soll sich die Turtle



Abbildung 23: Die ausführliche Darstellung der Skriptsteuerung

drehen. Und zwar um so viele Grad, dass sie nach dem Ende der *ganzen* Prozedur wieder so steht wie zu Beginn.

Die Abbildung 24 auf der nächsten Seite zeigt zum Beispiel ein 10-Eck, das selbst zehnmal gezeichnet wurde. Vor jeder Zeichnung hat sich die Turtle um 36 Grad gedreht.

Machen wir uns an die Arbeit. Wir nennen unsere Ellipse für dieses Beispiel »Turtle«.

Kümmern wir uns zunächst um das Zeichnen eines einzigen *n-Ecks*. Erzeuge ein leeres Skript und nenne es *nEck*. Anders als bisher soll die Ausführung des Skriptes von einer übergebenen Anzahl der Ecken abhängig sein. Wir erzeugen über das Kontextmenü des Skriptes dafür einen so genannten *Parameter*. Wie war das noch mit dem Kontextmenü? Du musst auf den Namen »Turtle« (blauer Rand) in Deinem Skript klicken. Dort findest Du einen Menüeintrag, der das Einrichten eines Parameters ermöglicht.



Dieser Parameter `Turtle nEck:` ist rot umrandet. Die Bezeichnung *Number* ist dabei kein Name, sondern ein *Typ*. Der schwarze Pfeil zeigt Dir alle anderen Typen, die noch möglich sind. Wir benötigen aber eine Zahl, so dass *Turtle nEck: 6* zum Beispiel ein Sechseck zeichnen kann. Die Abbildung 25 auf der nächsten Seite zeigt Dir das fertige Skript. Wir benötigen offenbar eine Variable mit dem Namen *zaehler*. Diese kontrolliert die Anzahl der Durchläufe. Durch Anklicken des Parameters erzeugst Du eine Kachel mit der Aufschrift *Number*. Weitere Variablen sind *seite* und *winkel*. Als letzten Eintrag in der Test-Kachel siehst Du, dass sich *nEck* selbst aufruft. Allerdings mit dem um 1 erniedrigten *zaehler*.

Ganz ähnlich sieht das Skript für das Zeichnen der »Rosette« aus, wie Du in der Abbildung 26 auf Seite 25 erkennen kannst. Dieses Skript benötigt allerdings seinen eigenen *rosettenZaehler*, sowie den *rosettenWinkel* jeweils als neue Variablen.

Das dritte Skript heißt wieder *starten* (siehe Abbildung 27 auf Seite 25). Hier werden sämtliche Eigenschaften festgelegt. Alle Informationen über das Zeichnen einer Turtle findest Du in der Kategorie *Stifte*. Die Variablen *ecken* und *rosette* geben die beiden wichtigsten Parameter an. Als letzte Kachel siehst Du, die Nachricht `Turtle rosette:Turtles rosette`. Das ist kein Wortspiel, sondern ein sehr wichtiger, aber feiner Unterschied. Das erste Wort *Turtle* ist der Name des Empfängerobjekts. *rosette:* ist der Nachrichtenselektor. Das übergebene Argument heißt *Turtles rosette*.

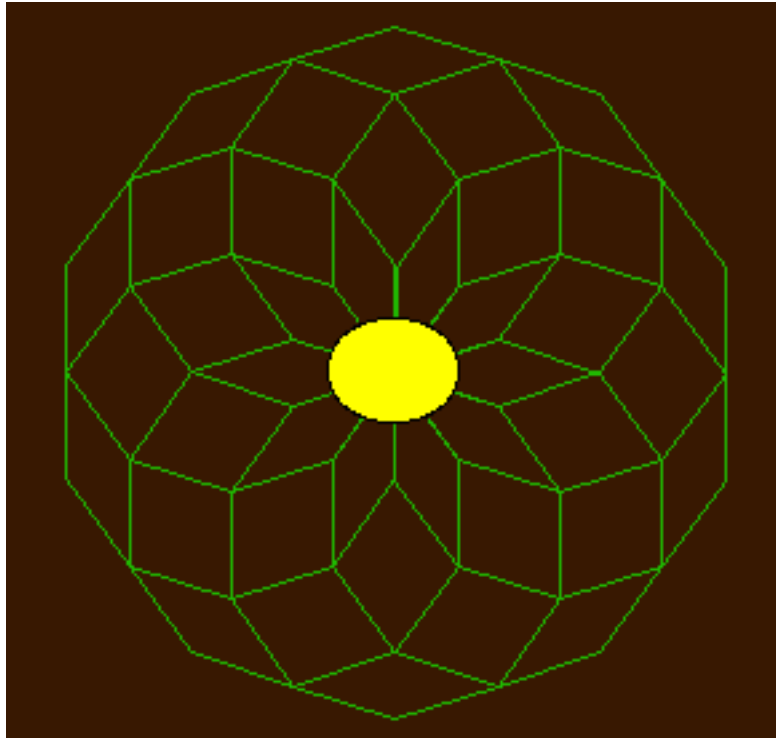


Abbildung 24: Eine Rosette mit eToys

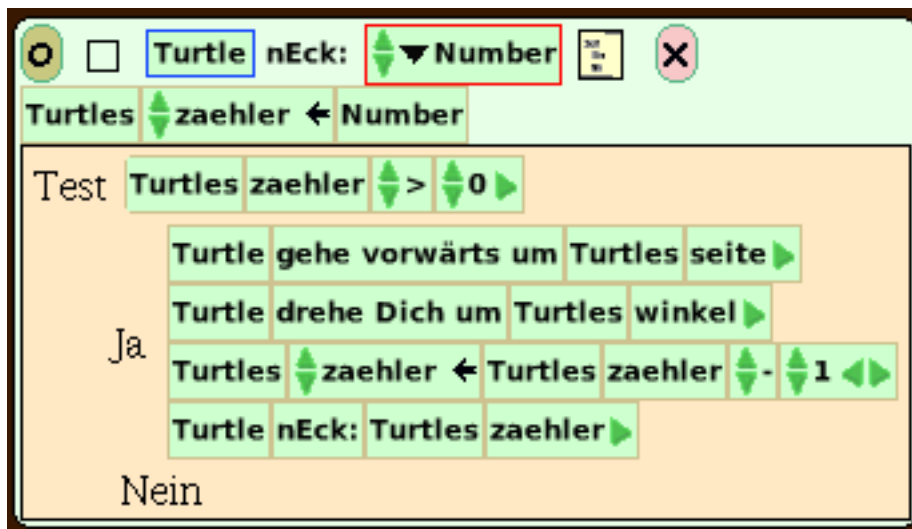


Abbildung 25: Zeichnen eines n-Ecks



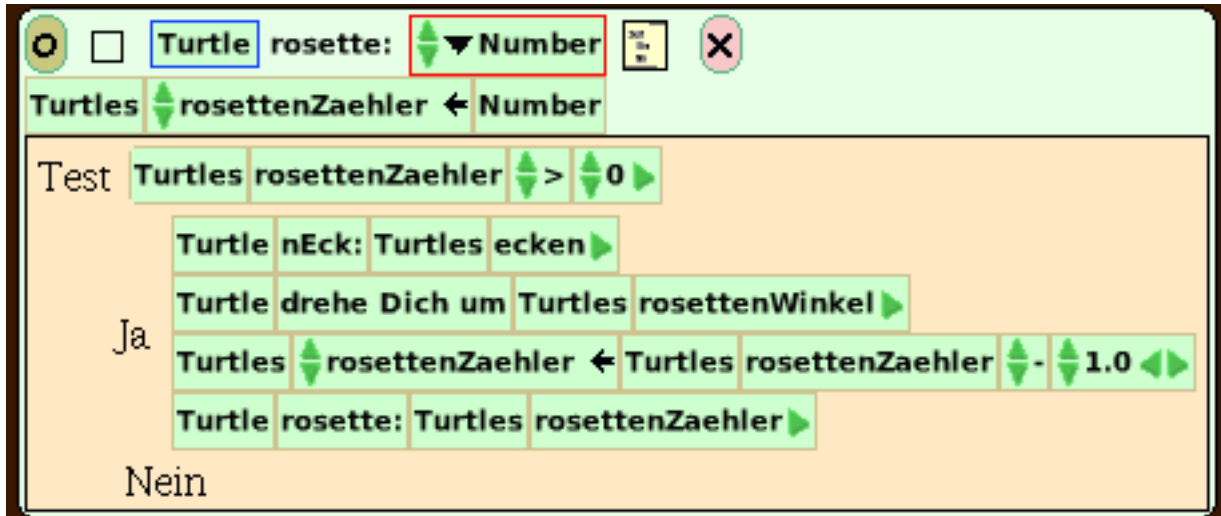


Abbildung 26: Zeichnen einer Rosette mit eToys

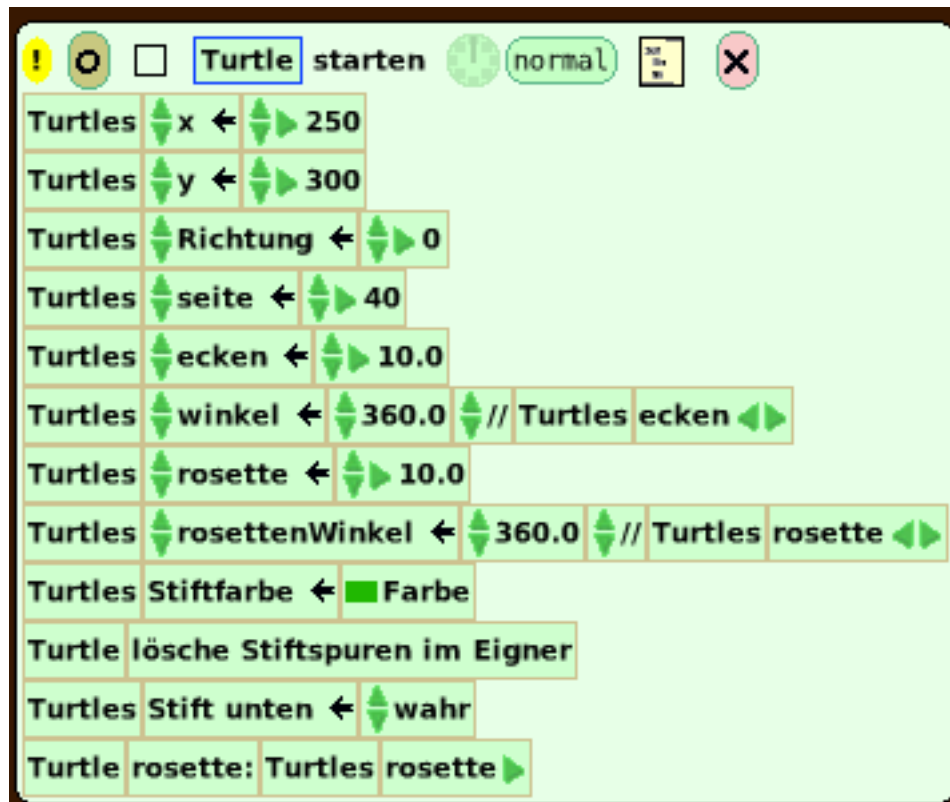


Abbildung 27: Startskript für die Rosette

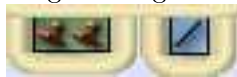
Falls Du keine Lust hast, die Skripte selbst zu erstellen, findest Du sie in *eToysTurtle.pr*. Dieses Projekt befindet sich ebenfalls in dem von Dir entpackten Archiv. Der Hintergrund dieses Beispiels ist schwarz. Ja. Natürlich ist auch das Projektfenster ein Morph, den Du verändern kannst. Achte darauf, dass Du die Seitenlänge nicht so groß wählst, dass die Turtle über den Rand zeichnen muss.

### 3.11 Zeichnen eines eigenen Morphs

Der Höhepunkt unserer kreativen Tätigkeit ist nun endlich das eigene Zeichnen eines Morphs. Den Malkasten findest Du in der blauen Lasche mit der Aufschrift *Geräte*. Das Icon dieses Wunderwerkzeugs ist die Farbpalette



Lässt Du diese auf dem Desktop fallen, öffnet sich eine Zeichenebene mit einem grafischen Werkzeug, das wirklich selbsterklärend ist. Jede Anleitung würde wahrscheinlich an dieser Stelle stören. Du kannst nach Herzenslust auf Entdeckungsreise gehen. Eine Sache allerdings wird gerne übersehen: Die etwas unscheinbaren Reiter, die Du in dieser Abbildung



siehst, sind sehr wichtig. Öffne sie doch einmal. Was findet sich da nicht alles? Und vor allem: Was bedeuten die einzelnen Stempel? Es ist wirklich nicht schwierig, das herauszufinden.

Die roten Knöpfe sind keine anklickbaren Buttons, sondern Schieber. Bei dem Rotations-



button ist zu bedenken, dass durch ihn die »Richtung 0« festgelegt wird. Wenn Du also eine Ellipse



mit diesem Button drehst, so wird sie sich bei dem Selektor *gehe vorwärts um* genau so nach oben bewegen.

Der Morph wird erst erstellt, wenn Du Deine Zeichnung mit OK bestätigst hast. Dann existiert Deine Zeichnung wie jeder andere Morph in der Squeak-Welt.

## 4 Steuern mehrerer Morphe: Insekten am Seerosenteich

Bisher haben wir eigentlich wenig mehr als herkömmliche Turtle-Grafik betrieben. In der Natur tauschen aber eine Vielzahl von Lebewesen Nachrichten mit einander aus. In diesem Abschnitt werden wir das Leben von einigen Insekten auf einer feuchten Wiese mit einem Seerosenteich simulieren. Dafür benötigen wir zunächst nur eine Libelle *leila*.



Öffne ein Projektfenster. Natürlich zeichnest Du *leila* mit dem Malkasten. Die beiden Enden der Fühler sind ausgefüllte Kreise mit leicht unterschiedlichen Farben. Das wird nachher wichtig werden, wenn *leila* über den Teich fliegt. Aber diesen Teich zeichnen wir vorerst noch nicht. Die »Wiese« ist nur der eingefärbte Hintergrund des Projektfensters. Du erinnerst Dich? Auch ein Projektfenster ist ein Morph und verfügt über einen Halo. Dort gibt es ebenfalls den pinkfarbenen Button für das Ändern der Farbe. Oder den Viewer. Denke daran, dass es besser ist, *leila* groß zu zeichnen. Du kannst sie hinterher über den Halo verkleinern.

Du merkst an der Art dieses Textes, dass wir die aus den vorherigen Abschnitten bekannten Details nicht mehr genauer erläutern. Natürlich ist das beabsichtigt. Dieser Abschnitt bringt Dir zwar auch wesentlich neue Elemente bei. Aber er führt Dich auch zu selbstständigem Arbeiten. Die Beschreibungen beschränken sich daher immer mehr auf das eigentliche Vorhaben; also auf das *Was*. Sie erläutern nicht mehr so intensiv die Art der Umsetzung; also das *Wie*.

Falls Du vor »unlösbare« Probleme gerätst, können Dir die mitgelieferten Projekte mit dem Namen *eToysLibellenTeich* hoffentlich helfen. Versuche aber so viel wie möglich ohne Hilfe zu lösen.

### 4.1 Steuern eines Morpchs durch einen anderen

Wir wollen zunächst einmal etwas »Verrücktes« kennenlernen. Wir zeichnen uns ein Lenkrad



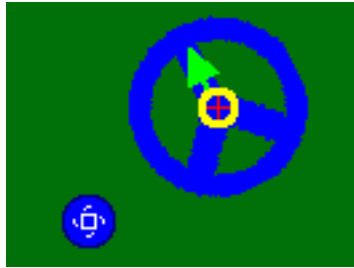
und werden den Flug von *leila* damit steuern. Das sieht mehr nach Technik aus. Als ob *leila* ein Flugzeug wäre. Richtig. Es passt auch nicht in unsere Seerosen-Welt. Aber es gehört natürlich in den Abschnitt »Steuern mehrerer Morphe«.

Welche Methoden benötigen wir nun? Erstaunlicherweise ist es nur eine. *leila* muss sich bewegen können. Aber die *Richtung* ihrer Bewegung soll der aktuellen Richtung des Lenkrads entsprechen. *leilas Richtung* muss also *lenkrads Richtung* zugewiesen werden. Das Skript

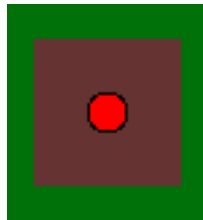
*leila bewegeDich* besteht also nur aus zwei Zeilen: Der Zuweisung von *lenkrads Richtung* zu *leilas Richtung* und der Nachricht *leila gehe vorwärts um...*. Lasse für den Anfang die Schrittweite von *leila* möglichst klein.

Wir werden im Text jetzt weitestgehend das Abdrucken der Skripte vermeiden. Du solltest *wirklich* alles selbst versuchen und nur bei Bedarf in dem Projekt *eToysLibellenTeich1.pr* nachsehen.

Starte nun das Skript *leila bewegeDich*. Du steuerst nun *leilas* Richtung über den blauen Rotationsbutton des Halos vom *lenkrad*.



Wenn es das erste Mal funktioniert, ist die Begeisterung sicherlich nicht allzu klein. Aber es gibt noch etwas viel Besseres. Das Lenkrad kann nur die Richtung steuern. Es kann aber nicht die Geschwindigkeit regeln. Über diese Fähigkeiten verfügt aber der *Joystick*. Du findest ihn in der roten Lasche mit der Aufschrift »Lager«.



Die Fähigkeiten dieses Tools werden gerne übersehen. Den roten Punkt kannst Du mit der Maus bewegen. Nach dem Loslassen läuft er aber immer wieder in die mittlere Position zurück. In dem Viewer gibt es nun eine eigene Kategorie mit dem Namen *Joystick*. Für uns sind die Kacheln *Joysticks Betrag* und *Joysticks Winkel* besonders interessant.

Ersetze zunächst in Deinem Skript in der Zuweisung zu *leilas Richtung* den Eintrag *lenkrads Richtung* durch *Joysticks Winkel*. Genauer gesagt: Du musst die Kachel neu schreiben. Ziehe die alte Kachel aus dem Skript heraus. Falls Du sie nicht mehr benötigst, wirf sie in den Mülleimer. Probiere dann danach das Skript aus, um ein Gefühl für die Steuerung mit dem Joystick zu gewinnen.

Nun soll auch *leilas* Geschwindigkeit über den Joystick gesteuert werden. Die Kachel *Joysticks Betrag* mag auf den ersten Blick nicht ganz verständlich sein. Gemeint ist dabei die Entfernung von der mittleren Stellung. Verändere jetzt die Kachel *leila gehe vorwärts um...* <irgendeine Zahl> in *leila gehe vorwärts um Joysticks Betrag*. Nach dem Starten bewegt sich *leila* natürlich noch nicht. Bewege nun den Joystick und versuche ein Gefühl für die feine Steuerung von *leilas* Geschwindigkeit und *leilas* Betrag zu bekommen.

In dem Projekt *eToysLibellenTeich1.pr* gibt es zwei Skripte mit den Namen *leila bewegeDichMitLenkrad* und *leila bewegeDichMitJoystick*. Du startest beide Skripte über die Skriptsteuerung durch Anklicken der Uhr. Bedenke allerdings beim Stoppen, dass Du mit *go* alle

angehaltenen Skripte startest. Daher solltest Du ein gestopptes Skript immer wieder in den Zustand *Normal* versetzen. Wenn die Skripte in der Skriptsteuerung nicht zu sehen sind, deaktiviere das Feld *nur laufende Skripte*.

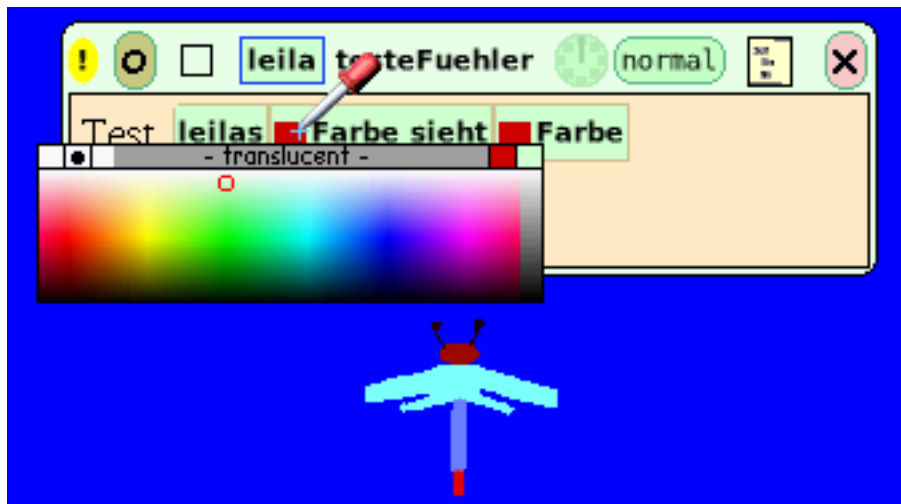
## 4.2 Abgleichen der Farbe

Wir entfernen nun aus unserem Projekt alle technischen Dinge. Also das *lenkrad*, den *Joy-stick* und auch das eine oder die beiden Skripte zur Bewegung. Nun beschäftigen wir uns nur noch mit der Natur. Unsere nächste Aufgabe besteht aus zwei Teilen.

1. Wir zeichnen einen Teich (noch ohne Seerosen) und eine hellgrüne Uferzone.
2. Leila bewegt sich etwa aus der Mitte des Teiches geradlinig vom Teich fort. Sobald sie mit dem linken Fühler die Uferzone verläßt, soll sie sich um 160 Grad nach links drehen und zurückfliegen. Sobald sie mit dem rechten Fühler die Uferzone verläßt, soll sie sich um 160 Grad nach rechts drehen und zurückfliegen.

Für den ersten Teil zeichne erst eine große hellgrüne Fläche. Diese Fläche sollte irgendeine rundliche Form besitzen, aber keine gleichmäßigen Konturen aufweisen. Nun gibt es zwei Möglichkeiten. Zeichnest Du *in* diese Fläche den blauen Teich hinein, so hast Du am Ende *einen einzigen* Morph. Teich und Uferzone gehören dann zusammen. Ich habe es vorgezogen, den Teich als separaten Morph zu zeichnen. Er wird dann im Projektfenster einfach auf die hellgrüne Uferzone gelegt. Der Nachteil ist der, dass die Uferzone den Teich völlig überdeckt, wenn sie angeklickt wird. Über das Menü (rotes Handle) kannst Du diesen Morph wieder nach hinten setzen. Über dasselbe Menü kannst Du auch bei Bedarf den Teich in die Uferzone einbetten, so dass ein Morph entsteht. Es lohnt sich, bei Gelegenheit das Menü hinter dem roten Handle einmal genauer zu studieren. Nenne bitte den Morph für das Ufer auch *ufer* und den Teich – na, rate mal – *teich* wäre als Name eine gute Idee.

Nun geben wir *leila* ihr Skript *leila bewegeDich*. Die Aktionen von *leila* bestehen allerdings aus zwei Teilen. Einerseits soll sich *leila* geradlinig bewegen. Nach jedem Schritt soll sie aber testen, ob einer ihrer Fühler die Uferzone verläßt. Mit anderen Worten: Es soll getestet werden, ob einer ihrer Fühler die Farbe des Hintergrundes sieht.



Wie wir ein Skript *leila testeFuehler* mit einer Testkachel erzeugen, wissen wir ja bereits. Wir haben aber noch nicht über die sehr wichtige Kategorie *Test* gesprochen. Dort findest Du bei jedem Morph sehr viele interessante Testmöglichkeiten. Eine davon heißt *leilas Farbe sieht*. Erzeuge nun das Skript *leila testeFuehler* mit der Testkachel und dem Eintrag *leilas Farbe sieht*. Die Kachel verändert sich beim Herausziehen aus dem Viewer in *leilas □Farbe sieht □Farbe*. Statt der Felder □ ist eine Farbe eingetragen. Klickst Du auf eines dieser Felder öffnet sich ein Farbspektrum mit der Pipette. Natürlich wählst Du keine Farbe aus der Farbpalette. Du bewegst stattdessen die Pipette auf den kleinen dunklen Punkt am linken Fühler von *leila*. Ist der Punkt *zu* klein, musst Du *leila* notfalls vergrößern. Die zweite Farbe ist natürlich das Dunkelgrün des Hintergrunds. Sollte diese Abfrage richtig sein, soll sich *leila* bekanntlich nach links um 160 Grad drehen. Füge die entsprechende Kachel in die Testkachel ein. Verfahre mit dem rechten Fühler in einer *zweiten* Kachel in *demselben* Skript entsprechend.

Nun benötigen wir nur noch das Skript *leila bewegeDich*. Dieses besteht aus der üblichen Bewegungskachel. Als zweite Kachel soll das Skript *leila testeFuehler* lediglich einmal ausgeführt werden. Setze den Status des Skriptes *leila bewegeDich* auf *angehalten*. Wegen des geringen Platzes solltest Du die Skriptsteuerung nicht in der ausführlichen Form aufklappen. Mit den Einstellungen 8 Ticks pro Sekunde bei einer Schrittweite von 5 bewegt sich *leila* immer noch wie ein Flugzeug. Wenn Du die Anzahl der Ticks auf 100 erhöhst, sieht das Ganze schon besser aus.

Bist Du mit dem Ergebnis zufrieden? Nun, je nach Uferzone kann noch etwas nicht so ablaufen wie wir es uns wünschen. Fliegt *leila* nicht mehr über den Teich, sondern entlang einer breiteren Stelle der Uferzone nahezu parallel zu ihm, könnte es sein, dass sie doch aus der Uferzone ausbricht. Natürlich könnten wir den Drehwinkel auf 180 Grad setzen. Dann fliegt *leila* ständig hin und her. Es gibt aber noch eine viel bessere Möglichkeit.

Das Projekt für dieses Kapitel heißt *eToysLibellenTeich2.pr*.

### 4.3 Zulaufen auf einen bestimmten Punkt


Unser Problem versuchen wir nun eleganter zu lösen. Auch der Teich – so groß wie er ist – besitzt eine bestimmte Position auf dem Bildschirm. Genauer gesagt: es ist der Mittelpunkt des Teiches, der diese Position besitzt. In der Kategorie *Bewegung* von *leilas* Viewer findest Du die Kachel *leila turn toward Punkt*. Der Teil *Punkt* bezeichnet nun ein allgemeines Objekt, dass natürlich durch die Kachel des Teiches ersetzt werden muss<sup>3</sup>.

Ändern wir nun unsere beiden Test-Kacheln ab. Statt der Drehung setzen wir die Kachel *leila gehe in Richtung Punkt* ein. Über das orangefarbene Handle des Halos von *teich* kannst Du eine Kachel mit dem entsprechenden Namen erzeugen und damit *Punkt* ersetzen. Nun wird die Bewegung allerdings etwas langweilig. Dann nach dem Verlassen der Uferzone dreht sich *leila* immer zum selben Punkt. Wir wollen nun dafür sorgen, dass *leila* nicht *genau* zu dem Mittelpunkt des Teiches läuft, sondern maximal 10 Grad von dieser Richtung abweicht. Wenn sie mit dem linken Fühler aus der Uferzone trat, soll sie um maximal 10 Grad nach links von dieser Linie abweichen. Linksdrehungen werden in Squeak – seltsam genug – durch *negative* Winkel vorgegeben. Das weicht bekanntlich von den Gepflogenheiten in der Mathematik ab.

Doch wir müssen auch noch klären was eigentlich *maximal* bedeuten soll. Die Lösung: Wir erzeugen eine »Zufallskachel«. Diese findest Du im Kontextmenü eines jeden Skriptes. Wie

<sup>3</sup>Die Kachel trug in der Version Squeak 3.8 noch eine englische Bezeichnung. Das kann in einer neueren Version bereits geändert sein.

war das noch? Klicke auf den blau umrandeten Namen von *leila* im Skript *leila testeFuehler*.

Du erhältst eine Kachel, die so aussieht: . Wahrscheinlich steht als Zufallszahl bei Dir 180. Ändere diese Zahl auf 10. In dem Beispielprojekt *eToysLibellenTeich3.pr* habe ich 16 gewählt. Wie groß die Abweichung von der Richtung zum Teich werden kann, hängt von der Gestalt Deiner Uferzone ab. Baue jetzt zwei entsprechende Zuweisungen für *leilas Richtung* in das Skript *leila testeFuehler* in die Test-Kachel ein.

Jetzt sollte *leila* über den Teich schwirren, ohne aus dem Bild zu laufen. *leila* verläßt die Uferzone unterschiedlich weit, bevor sie umkehrt. Denke daran, dass diese Entfernung auch von der Schrittweite abhängt.

Das Beispiel findest Du, wie gesagt, in der Datei *eToysLibellenTeich3.pr*.

#### 4.4 Erscheinen und Verstecken eines Morphs

Kümmern wir uns nun um die Seerosen. Unsere Seerose soll in vier verschiedenen Zuständen vorkommen: *Versteckt* (unter der Wasseroberfläche), *Knospe*, *völlig entfaltete Blüte* und *verwelkte Blüte*. So verrückt es Dir auch erscheinen mag: Unsere Seerose ist zunächst ein ganz *beliebiger* Morph. Zum Beispiel ein einfacher, ausgefüllter Kreis. Nenne diesen Morph *seerose*, auch wenn wirklich noch absolut nichts an eine Seerose erinnert. Die Knospe, die völlig entfaltete Blüte und die verwelkte Blüte sind später *eigene* Morphe. Unsere Seerose besitzt nun einen Zähler. Je nach Zählerstand soll sie das »Kostüm« eines dieser anderen drei Morphe tragen oder eben unsichtbar sein.

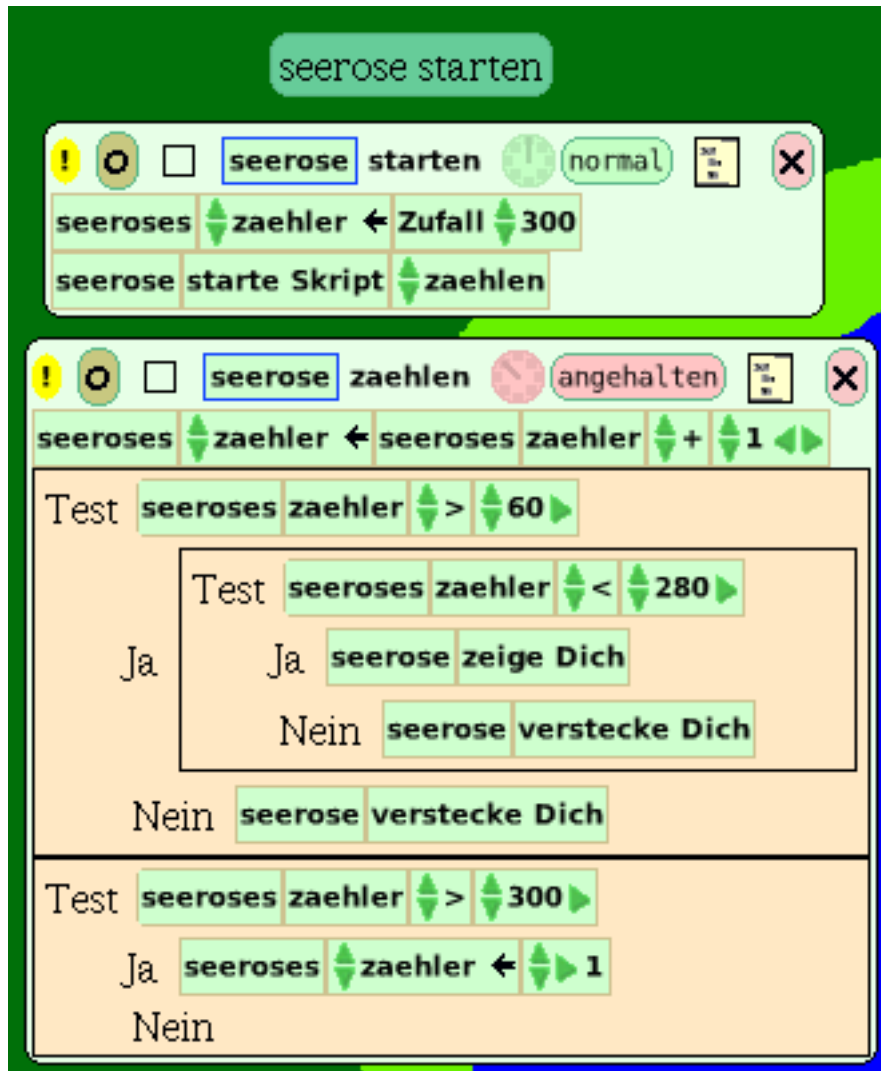
So weit unser Plan. Der seltsame Kreis ist also nur ein »Platzhalter« für die drei Kostüme der Seerose. Wir wollen den Kreis in diesem Abschnitt bei Bedarf verschwinden oder auftauchen lassen. Führe nun die folgenden Schritte aus:

**seeroses zaehler:** Diese Variable soll den Lebenslauf von *seerose* kontrollieren.

**seerose starten:** Dieses Skript setzt die Variable *seeroses zaehler* auf 0 und startet das Skript *seerose zaehlen* auf. Dieses Skript wird nur einmal angestoßen und verbleibt danach im Zustand »Normal«.

**seerose zaehlen:** Dieses Skript erfüllt drei Aufgaben. Erstens setzt es *seeroses zaehler* um 1 nach oben. Nur dann, wenn *seeroses zaehler* größer als 60 und kleiner als 280 ist, soll *seerose* sichtbar sein. Dafür benötigst Du die Kacheln *seerose zeige Dich* und *seerose verstecke Dich* aus der Kategorie »Verschiedenes«. Ferner soll dieses Skript *seeroses zaehler* zurücksetzen, wenn dieser den Wert von 300 überschreitet. Das Skript wird von *seerose starten* angestoßen und läuft, bis es von der Skriptkontrolle angehalten wird.

Führe jetzt diese drei Schritte aus. Wenn der Ablauf funktioniert (lasse auch gleichzeitig *leila* laufen), solltest Du im Skript *seerose starten* den Anfangsstand von *seerose zaehler* auf eine Zufallszahl zwischen 0 und 300 setzen.



Ausnahmsweise zeigen wir an dieser Stelle einmal die genaue Lösung der Aufgabe. Da es bei den Abfragen von Wahrheitswerten keine Und-Konstrukte gibt, muss ein verschachtelter Test erfolgen.

Das fertige Beispielprojekt für diesen Abschnitt heißt *eToysLibellenTeich4.pr*.

#### 4.5 Ein Morph trägt das Kostüm eines anderen

Wir zeichnen nun die drei Morphs, die den einzelnen Zuständen der Seerose entsprechen.



Dieser Morph heißt *knospe*. Die Knospe soll zwischen den Zählerständen 60 und 100 sichtbar sein.



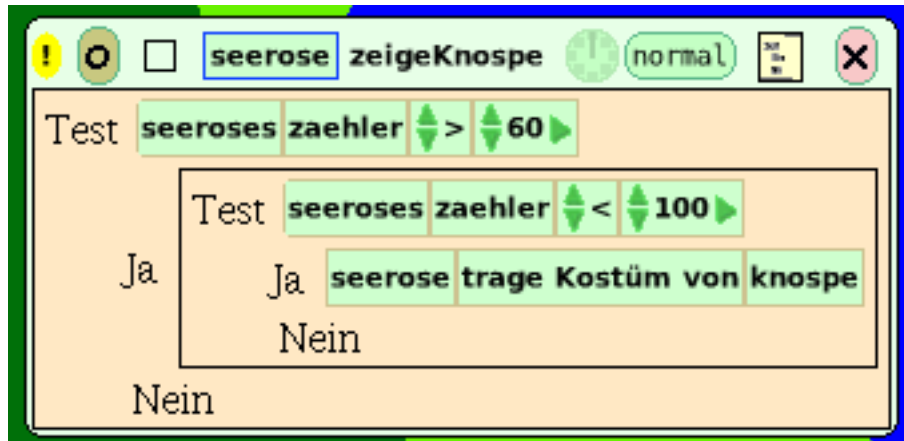
Dieser Morph heißt *bluete*. Die Blüte ist zwischen den Zählerständen 100 und 220 zu sehen.



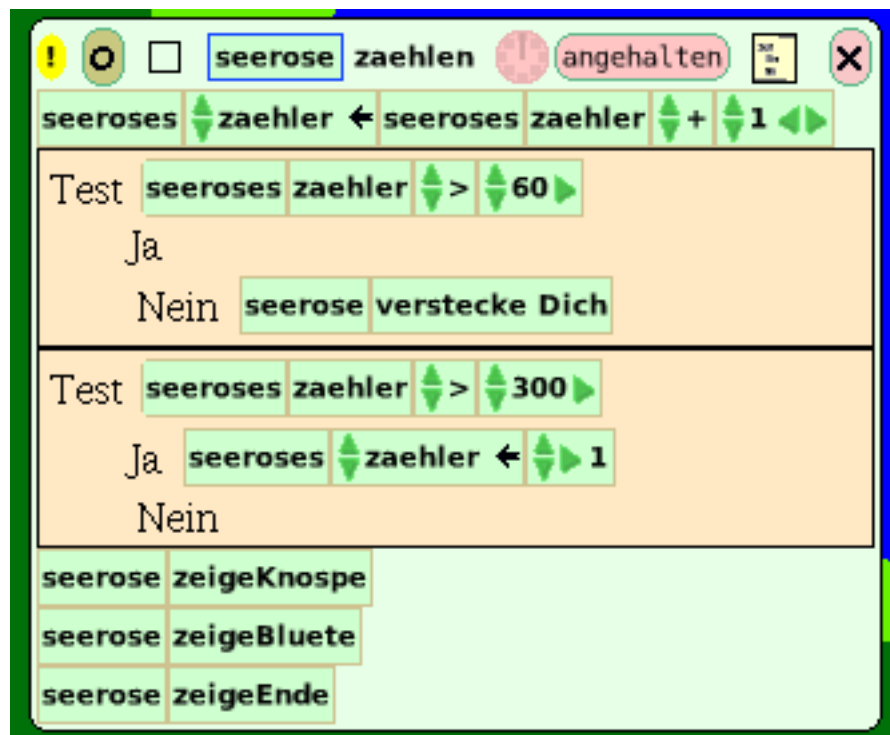


Und dieser Morph heißt *ende*. Der Endzustand einer verblühten Seerose ist zwischen den Zählerständen 220 und 280 zu sehen.

Unter der Kategorie »Verschiedenes« im Viewer der Seerose findest Du die Kachel *seerose trage Kostüm von Punkt*. Schreibe nun die Skripte *seerose zeigeKnospe*, *seerose zeigeBluete* und *seerose zeigeEnde*. Die folgende Abbildung zeigt das Skript für *seerose zeigeKnospe*:



Entsprechend richtest Du die anderen Skripte ein. Das Skript *seerose zaehlen* musst Du etwas abwandeln, wie die nachfolgende Abbildung zeigt:



Der eigentlich aktive Morph ist also nach wie vor *seerose*. Die anderen drei Morphpe dienen nur als Vorlagen. Du kannst sie daher minimieren, wenn Du sie nicht als eigenständige Morphpe benötigst. Dabei verschwinden sie nicht ganz, sondern sie zeigen sich als Icon

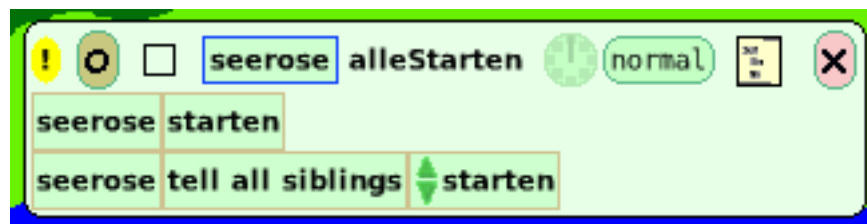


am oberen linken Rand des Projektfensters. Überzeuge Dich nun, dass alles so funktioniert wie es soll. Das Beispielprojekt ist in der Datei *eToysLibellenTeich5.pr* enthalten.

## 4.6 Siblings

Nun kannst Du beliebig viele Seerosen erzeugen. Du erinnerst Dich, dass Kopien über das grüne Handle vorgenommen werden. Einen Zwilling erhältst Du über das grüne Handle *ohne* die SHIFT-Taste zu drücken. Erzeuge Dir zwei Zwillingungskopien von *seerose*. Vielleicht auch noch eine Zwillingungskopie von *leila*. Entscheidend ist aber, dass die Kopien von *seerose* Zwillingungskopien sind.

Ein Morph kann nun alle seine Zwillingungskopien ansteuern. Du startest die *seerose* jetzt nicht mehr über *seerose starten*. Sondern Du schreibst ein neues Skript *seerose alleStarten*, das die folgenden Kacheln enthält:



Die letzte Kachel findest Du wieder in der Kategorie »Verschiedenes«. In dem Beispielprojekt *eToysLibellenTeich6.pr* ist die Zwillingungskopie von *leila* ein wenig verändert, so dass Du den Zwilling von dem Ursprungsmorph unterscheiden kannst.


An dieser Stelle hört die Anleitung zu diesem Projekt auf. Du hast jetzt alle Informationen, um nach Herzenslust dieses – doch eigentlich sehr schöne – Projekt auszubauen. Du kannst eine zusätzliche Art von Seerosen erstellen, vielleicht auch ein paar Fliegen oder Mücken, auf die die beiden Libellen Jagd machen. Es gibt sehr viele weitere Kacheln auszuprobieren. Deiner Phantasie sind keine Grenzen gesetzt. Vielleicht erstellst Du auch einen Schmetterling, der nach jedem Schritt sein Aussehen verändert, so dass Du wirklich glaubst, er würde über den Teich flattern . . .

## 5 Unterschiede zwischen eToys (*Self*) und Smalltalk\*


Dieser Abschnitt ist ausschließlich für Leser gedacht, die bereits über Smalltalk-Kenntnisse verfügen. Alle anderen können diesen Abschnitt überspringen.

Sehr wahrscheinlich sind Dir beim Umgang mit eToys viele bekannte Smalltalk-Elemente aufgefallen. Aber es gibt auch ganz entscheidende Unterschiede. Wir gehen der Einfachheit halber von einem Beispielmorph aus, der *Mairi* heißt.

**Namen:** Bei den Namen tauchen schon die ersten Unterschiede auf. Sie dürfen groß geschrieben werden. In Smalltalk würde dies zugleich eine globale Variable oder eine Klasse kennzeichnen. Morphe werden aber bei eToys strenggenommen nicht durch Klassen erzeugt, sondern durch Kopien eines so genannten *Prototyps*. Diese Idee geht auf einen in den frühen sechziger Jahren von Ivan SUTHERLAND erfundenen grafischen Editor zurück, der *Sketchpad* heißt. Noch lange, bevor objektorientiertes Denken überhaupt bekannt war, konnten mit diesem Editor grafische Prototypen gezeichnet werden, deren Kopien ein »Eigenleben« hatten und selbstständige Individuen darstellten. Es ist gewiß nicht übertrieben, wenn wir sagen, dass *Sketchpad* die Entstehung objektorientierten Denkens in entscheidendem Maße gefördert hat. Während es in Smalltalk – sehr konequent – *ausschließlich* nur Objekte gibt, geht der Smalltalk-Ableger *Self* zu der alten Idee der Prototypen zurück. Und *Self* ist die Sprache von eToys. In Java sind ebenfalls die Klassen *Prototypen* und keine wirklichen Objekte, denen Du Nachrichten schicken könntest. Insofern ist eigentlich *Self* die Mutter von Java, während *Smalltalk* eher als deren Großmutter zu betrachten ist.

**Zahlen:** Zahlen sind in *Self* – genau so wenig wie in Java – Objekte, sondern »primitive Datentypen«. Der einzige sichtbare Vorteil ist die damit verbundene Möglichkeit, Punkt vor Strichrechnung zu gewährleisten. Das war es aber schon. Der Nachteil einer solchen Auffassung ist ein ständiges Hin- und Her zwischen objektorientiertem Denken und halb prozeduralem Denken. Wir rechnen zwar auch im Alltag *mit* Zahlen. Manche extrem starken Konstrukte von Smalltalk, wie zum Beispiel *3 times:aBlock* fehlen daher. Bei eToys fehlen tatsächlich Zählschleifen. Wenn sie nicht über einen Dauerklick auf  in Skripten realisiert werden, musst Du sie über Zählvariablen und Selbstaufrufe eigenhändig zusammenbauen.

**Kacheln** Unter den Kacheln gibt es zwei verschiedene Arten:

**Kacheln für Instanzvariablen:** Kacheln, die die Instanzvariablen verändern oder abfragen, fangen mit *Mairis* an. Der Zuweisungspfeil  wurde von Smalltalk übernommen. Kacheln wie *Mairis x* entsprechen dem Aufruf einer *get* Methode mit dem Namen *x*. Es ist nicht etwa die Variable *selbst*, die aufgerufen wird.

**Kacheln für Nachrichten:** Alle anderen Kacheln sind Nachrichten an *Mairi*. Mit anderen Worten: Alle in *Mairis* Viewer befindlichen Nachrichtenkacheln sind eigentlich Nachrichten an den Parameter *self* (daher der Name der Sprache!). Sehr problematisch wäre die naheliegende Sichtweise, dass Du als Programmierer *Mairis* Methoden in ein Skript einbaust. Das ist *nicht* der Fall! In dem Skript sendest Du *Mairi* Nachrichten. Aktiv ist nur *Mairi* selbst. Die problematische Auffassungsweise tritt auch bei Java auf. Eine Kachel wie *Mairi gehe vorwärts um 5* hätte die Gestalt *Mairi.geheVorwaertsUm(5)*. Auch hier sieht es nicht nach einer *Nachricht* an *Mairi* aus, sondern – völlig *falsch* – nach dem Aufruf einer Methode, die *Mairi* gehört.

**Schleifen:** In eToys sind die elementaren Schleifen kurioser Weise nur als Endlosschleifen über die Uhr realisiert. Der Abbruch einer solchen Schleife geschieht durch den Programmierer oder durch eine entsprechende Nachricht in einem Skript. Der Durchlauf der Schleife ist – einmal angestoßen – von der Abfrage eines Wahrheitswertes anschließend völlig *unabhängig*. Der gesteuerte Abbruch kommt sozusagen *von außen*.

**Skripte:** Skripte sind nun gewissermaßen so etwas wie *Blöcke* in Smalltalk. Ihre Ausführung wird allerdings nicht durch gezieltes Absetzen einer *value*-Nachricht eingeleitet,

sondern durch die Aktivierung eines *Dämons*, der sozusagen unabhängig vom aktuellen Geschehen das Skript ablaufen läßt.

Wir können unsere Gedanken mit der Feststellung zusammenfassen, dass eToys sozusagen eine einfachere Form von Smalltalk ist. Es gibt keinen »unnötigen« Luxus. Gewisse Dinge, wie etwa *oder* Konstrukte bei Tests wären aber sehr wünschenswert. Ein Test in eToys ist immer mit dem Abfragen eines *einzigsten* elementaren logischen Konstrukts verbunden. Sollen *und* und *oder* Abfragen durchgeführt werden, so müssen diese über mehrere Test realisiert werden.

Das Fehlen einer logischen Steuerung von Schleifen macht die Erstellung von Rekursionen schwierig und in manchen Fällen unmöglich. Das ist allerdings eine wesentliche Stärke der Turtle-Grafik, die als Mutter von eToys betrachtet werden muss. eToys ist ein einzigartiges Feature von Squeak, um Kinder an die Programmierung von Grafik heranzuführen. Sicherlich haben auch Profis damit ihren Spaß. Doch für große Projekte ist die Sprache zu elementar. Hier lotet Smalltalk erheblich tiefer.

## 6 Kleine Kontrollfragen

**Frage 1** *Welchen Vorteil hat die Verwendung von Kacheln statt die Verwendung von purem Smalltalk (oder Self)?*

**Frage 2** *Mit welchem eToy kannst Du sowohl die Geschwindigkeit als auch die Bewegungsrichtung eines Morphs steuern?*

**Frage 3** *Was bedeutet das graue Handle (mit einem Maulschlüssel gekennzeichnet) in dem Halo eines Morphs?*

**Frage 4** *Findest Du eine Möglichkeit, einen Morph exakt auf einem Kreis laufen zu lassen? Wie sieht es mit der Möglichkeit aus, die Bewegung des Mondes und der Erde um die Sonne zu simulieren? (Vorsicht: Diese Aufgabe gehört wahrscheinlich zu den schwierigeren)*

**Frage 5** *Welche grafische Erfindung ist sozusagen die Mutter von eToys?*

**Frage 6** *Wie heißt im Internet die beste Anlaufstelle für eToys?*



## Index

EToys, 4

Java, 35

Joystick, 28

Logo, 4

Morph, Duplikat, 6

Morph, Zwilling, 6

Papert, Seymour, 4, 22

Prototyp, 5, 35

Self, 13

sibling, 6

Sketchpad, 35

Skripte, 4

Sutherland, Ivan, 35

Turtle-Grafik, 4

Zwillingskopien, 34