

Der Mensch spielt mit!

Serie Realbasic, Folge 3 In diesem Teil spielen wir als Programmierer und Anwender endlich selbst mit. Bis es soweit ist, sollte unser Spiel etwas freundlicher werden *von Christian Schmitz*

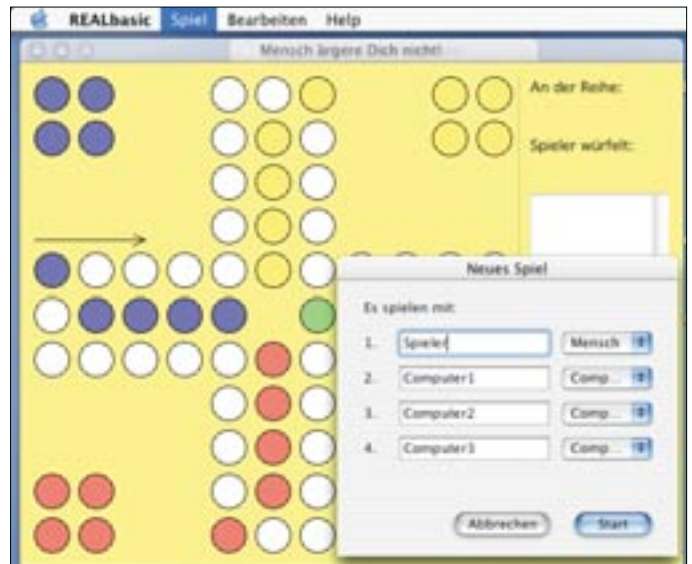
◆ **WIR BEGINNEN DAMIT**, die Menüleiste etwas aufzumöbeln. Wir hätten gerne einen Eintrag für eine neues Spiel, einen Eintrag für ein neues Spielbrett und einen Eintrag zum Schließen eines Fensters.

Wir öffnen das Menü im Projektfenster. Dort steht „File“ beziehungsweise „Ablage“ für das erste Menü. Für Programme, die keine Dateien erzeugen oder öffnen, empfiehlt Apple in den Human Interface Guidelines (Richtlinien für die Gestaltung von Benutzeroberflächen) dieses Menü anders zu nennen. Wir nennen es einfach „Spiel“, denn dort soll man ein neues Spiel starten können beziehungsweise ein Laufendes beenden.

Wir klicken auf „File“ bzw. „Ablage“ und in der Eigenschafts-Palette erscheinen die Eigenschaften des Menüs. Dort geben wir bei Text „Spiel“ ein. Die Konsequenz sehen wir direkt im Menüfenster.

Wir legen nun einen neuen Menüeintrag an. Dazu klicken wir unten im Menü auf den leeren Eintrag und tippen den Text „Neues Spiel...“ ein. Als Name sollte Realbasic automatisch „SpielNeuesSpiel“ eintragen. Sollten bei Ihrer Realbasic-Version Punkte am Namen auftauchen, müssen Sie diese auf jeden Fall entfernen, denn die Namen für Variablen dürfen keine Punkte enthalten. Beim Commandkey geben wir als Kurzbefehl „N“ ein. Bitte immer nur Großbuchstaben dort eingeben, denn bei Befehlskürzel unterscheidet man nicht zwischen Groß- und Kleinschrift.

Der nächste Menüeintrag wird „Neues Spielbrett...“ lauten und heißt im Code dann analog „SpielNeuesSpielbrett“. Im Spiel darf man ein Spielfenster jederzeit schließen und ein neues Spiel öffnen. Für dieses Menüitem tragen wir beim Commandkey „Shift-N“ ein. Der Benutzer muss also zusätzlich noch die Umschalttaste drücken. Falls wir mit einer englischen Realbasic-Version arbeiten, sollten



Nomen est omen In dieser Ausgabe programmieren wir unter anderem einen Einstellungsdialog, mit dem alle Spieler Namen bekommen.

wir nun alle Menüeinträge eindeutigen (Bearbeiten: Widerrufen, Ausschneiden, Kopieren, Einfügen, Löschen). Außerdem benennen wir das Spielfenster in „SpielFenster“ um. Zunächst ändern wir den Namen in der Eigenschafts-Palette und anschließend ersetzen wir ihn im gesamten Code. Dazu benutzen wir den Suchen-Dialog. Wir suchen nach „Window1.“ und ersetzen es durch „SpielFenster.“

Neues Spiel auf Wunsch

Ein neues Spiel soll starten, wenn man im Menü „Spiel“ den passenden Menüeintrag auswählt. Wir öffnen das Spielfenster im Codeeditor und dort den Open-Event. In diesem Event steht bereits die Zeile „NeuesSpiel“, die wir jetzt löschen.

Stattdessen legen wir einen neuen Menühandler für den Menüeintrag „SpielNeuesSpiel“ an. Dorthin schreiben wir „NeuesSpiel“ als Befehl und anschließend „Return true“ um Realbasic zu sagen, dass dieser Menüeintrag von uns bearbeitet wurde.

Im Event „EnableMenuItems“ müssen wir diesen Menüeintrag noch aktivieren. Dazu ergänzen wir dort die Zeile „SpielNeuesSpiel.enable“.

Nun kontrollieren wir noch einmal, ob alle Timer im Fenster auf Mode 0 stehen, denn sonst gibt es Laufzeitfehler im Programm. Damit man ein laufendes Spiel schließen kann, legen wir einen weiteren Menühandler für

EINSTIEG

Realbasic ist eine einfach zu erlernende Programmiersprache, in der auch Anfänger ihre ersten Programmiererfahrungen sammeln können. Mit dieser Serie können Sie ein Brettspiel à la „Mensch ärgere Dich nicht“ programmieren. In dieser Folge binden wir den Anwender als Spielpartner ein.

Serie: Realbasics für Profis

- | | |
|---------------------------|--------------|
| 1 Interface | Heft 12/2001 |
| 2 Computerspieler | Heft 01/2002 |
| 3 Mensch spielt mit | Heft 02/2002 |
| 4 Netzwerk | Heft 03/2002 |
| 5 Kompilieren und Release | Heft 04/2002 |

„SpielSchliessen“ an und ergänzen die Zeilen „Close“ und „Return true“. Im EnableMenuItems-Event schreiben wir „SpielSchliessen.enable“.

Neues Fenster

Für die Menübefehle, die auch ohne Fenster funktionieren sollen, benötigen wir eine Klasse vom Typ „Application“ in der wir die Menühandler versorgen. Dazu legen wir eine Klasse an, geben ihr den Namen „App“ und wählen als Superklasse „Application“ aus. Im NewDocument-Event der Klasse legen wir ein neues Spielbrett mit den folgenden Zeilen an:

```
Dim f as SpielFenster
F = New SpielFenster
```

Starten wir nun das Programm, entdecken wir zwei geöffnete Spielbrett-Fenster, da in den Projekteinstellungen das Spielfenster als Default ausgewählt ist. Das ist jetzt nicht mehr nötig, weshalb wir dort „none“ auswählen.

Global und lokal

Nun müssen wir noch eine größere Änderung vornehmen, denn an mehreren Stellen im Code sprechen wir das Fenster über den Namen „Spielfenster“ an. Das funktioniert gut, solange es nur ein Fenster gibt aber bei mehreren Fenstern ist es nicht eindeutig. Realbasic legt beim Zugriff auf ein Fenster über dessen Namen automatisch dieses Fenster an, weshalb wir bisher problemlos Methoden wie SpielFenster.NextFeld aufrufen können. Um aber gleichzeitig mit mehreren Fenstern spielen zu können, stattdessen wir einige Objekte mit Verweisen auf ihr eigenes Fenster aus.

In der Spieler-Klasse legen wir dazu eine neue Eigenschaft „Fenster as SpielFenster“ an und ändern alle Zugriffe auf das Spielfenster innerhalb dieser Klasse um auf „Fenster“. Der Constructor Spieler bekommt einen zusätzlichen Parameter f mit dem Verweis auf das Spielfenster. Die Parameterzeile sieht dann so aus: „n as String, c as Color, o as integer, f as SpielFenster“. Mit der Zeile „Fenster = f“ speichern wir den Verweis für später. In unserem Spielfenster ändert sich dadurch in der Methode „NeuesSpiel“ der Aufruf des Constructors für die neuen Spieler. Ein „self“ als weiterer Parameter übergibt dem Spieler bei seiner Erschaffung einen Verweis auf sein Fenster.

Zwei Spiele nebeneinander

Um ein neues Spielbrett anzulegen, erzeugen wir in unserer App-Subklasse einen neuen Menühandler „SpielNeuesSpielbrett“ mit folgendem Inhalt:

```
NewDocument
Return true
```

Tip | Ideen für den Ausbau der Simulation

Im momentanen Zustand ist das Spiel zwar spielbar, aber man könnte einiges verbessern, was jedoch den Rahmen unserer Serie sprengen würde. Hier ein paar Anregungen:

- Unsere Beispiele sind mit einer Uhr als nette Dekoration ausgestattet. (Tipp: das Date Objekt liefert die aktuelle Zeit)
- Man könnte die Spielfarben frei wählbar machen. (Tipp: der SelectColor-Befehl hilft)
- Die Computergegner könnten zufällig Namen aus einer (editierbaren) Liste von Namen bekommen. (Tipp: Rnd- und nthField-Funktion helfen)
- Alle Einstellungen wie die Spielernamen könnte man für das nächste Spiel in einer Preferencesdatei speichern (Tipp: BinaryStream über ein Child-Folderitem vom PreferencesFolderitem)
- Das Spiel ist nicht 100% nach den Regeln. Am Ziel kommt man mit jeder möglichen Würfelzahl ins Häuschen. Ändern Sie das ab, so dass es nur bei passender Würfelzahl funktioniert.
- Den Würfelvorgang könnte man grafisch als eine Art Glücksrad gestalten, das man per Maus anhält oder das einfach langsamer wird.
- Eine Anzeige der möglichen Züge und deren Risiken beziehungsweise Vorteile. Ähnlich zu den Tipps im Schachprogramm von Apple.

Wir rufen hier lediglich den Event „NewDocument“ auf, der dann eigenständig das neue Fenster anlegt.

Ein Bug bei NeuesSpiel!

Lassen wir das Spiel nun laufen, fällt uns beim Neustart eines Spiels ein kleiner Bug auf. Die „alten“ Figuren auf dem Spielbrett werden nicht entfernt, bevor das neue Spiel startet. Also ergänzen wir in der Feld-Klasse die Methode „Clear“ die folgenden Code beinhaltet:

```
Figur = Nil
Redraw
```

Hierdurch löschen wir den Verweis zur Figur und zeichnen das Feld neu. In der Methode „NeuesSpiel“ ergänzen wir zudem:

```
For i=0 to 3
  Brett(i).Clear
Next
For i=0 to 3
  yZiel(i).Clear
  rZiel(i).Clear
  gZiel(i).Clear
  bZiel(i).Clear
Next
```

Damit haben wir den Bug beseitigt.

Jedem Spieler seinen Namen

Jeder Spieler sollte einen eigenen Namen bekommen. Dazu konstruieren wir ein spezielles Dialogfenster. Wir benötigen in diesem Fenster vier Eingabefelder für die Namen der vier Spieler und dahinter jeweils ein Pop-up-Menü für „Mensch“ oder „Computer“. Das Ganze

sollte mit Texten (und eventuell auch Bildern) geschmückt werden. Rechts platzieren wir einen Knopf mit der Beschriftung „Start“. Links daneben einen „Abbrechen“-Knopf. Im Codefenster dieses Dialogs ergänzen wir eine Eigenschaft „Okay as Boolean“. Der Code für den Action Event vom OkayButton lautet:

```
Okay = true
Hide
```

Beim Abbrechen-Knopf schreiben wir:

```
Okay = false
Hide
```

In beiden Fällen speichern wir zunächst, ob der Anwender den Okay-Knopf gedrückt hat und blenden den Dialog dann wieder aus. Würden wir den Dialog ganz schließen (Close), wüssten wir später nicht mehr, ob die Variable Okay true oder false ist. Genau genommen schließen wir diesen Dialog erst beim Programmende, denn wenn wir ihn beim nächsten Aufruf von NeuesSpiel wieder zeigen, besitzt er immer noch alle Eingaben vom letzten Mal. Wir benennen unser neues Dialogfenster „NeuesSpielDialog“.

Der Eingabedialog verursacht ein paar Probleme

Die Abfrage der Spielernamen gehört in die Methode „NeuesSpiel“. Wir setzen dort direkt hinter den Dim-Zeilen die Zeile „NeuesSpielDialog.showmodal“. Dadurch erscheint der Dialog. Spätestens beim zweiten Mal sieht man, dass das Spiel im Hintergrund weiter läuft. Wir müssen also vor dem Aufruf des Di- ▶

alogs das Spiel anhalten. Die zwei verantwortlichen Timer halten wir mit folgenden zwei kurzen Code-Zeilen an:

```
Timer1.Mode = 0
Timer2.Mode = 0
```

Weiter unten stehen folgende zwei Zeilen, die wir direkt hinter dem Aufruf des Dialogs benötigen. Indem man den Modus eines Timers auf 2 setzt, startet man ihn und genau das brauchen wir, wenn der Dialog wieder verschwindet.

```
Timer1.Mode = 2
Timer2.Mode = 2
```

Das alte oder das neue Spiel läuft nun weiter, unabhängig davon, ob der Anwender „Ok“ oder „Abbrechen“ drückt. Beim Abbruchknopf sollte jedoch kein neues Spiel starten, sondern das alte weiterlaufen. Wir prüfen also die gespeicherte Okay-Eigenschaft des Dialogs und beenden die Methode mit Return, falls Abbrechen gedrückt wird:

```
If Not NeuesSpielDialog.Okay Then
    Return
End If
```

Damit stoßen wir wieder auf ein kleines Problem: Die Timer werden auch dann aktiviert, wenn noch gar kein Spiel läuft, was später zu Laufzeitfehlern führt. Wir müssen also feststellen, ob der Anwender das erste Spiel starten will oder ob er einen Spielneustart auslöst. Dazu definieren wir in dieser Methode eine Variable „ErstesSpiel“ als Boolean mit: „Dim ErstesSpiel as Boolean“.

Bevor wir die Timer auf 0 setzen, sollten wir über die Zeile „ErstesSpiel = Timer1.mode = 0“ den aktuellen Timerzustand abfragen und in dieser Booleschen Variable sichern. Kommt True heraus, dann lief vorher kein Timer und damit auch kein Spiel.

Nun müssen wir noch die Zeilen, mit denen wir die Timer starten, ändern, damit sie beim ersten Spiel nicht aufgerufen werden. Aber Achtung! Wenn wirklich ein Spiel gestartet werden soll, dann müssen die Timer auch

laufen, weshalb die folgende Bedingung etwas komplizierter ist: Auf Deutsch lautet die Bedingung: „Wenn es nicht das erste Spiel ist oder ein Spiel gestartet werden soll, dann“. In Realbasic sieht das so aus:

```
If Not ErstesSpiel or
NeuesSpielDialog.Okay Then
    Timer1.Mode = 2
    Timer2.Mode = 2
End If
```

Die Namen ins Spiel übernehmen

Ganz unten in dieser Methode sehen wir noch einige Zeilen aus der letzten Folge. Diese ändern wir durch folgenden Code so ab, dass sie die Spielnamen aus dem Dialog übernehmen.

```
Spieler(0) = New Spieler(NewesSpielDialog.Editfield1.text, rgb(0,0,255), 0, self)
Spieler(1) = New Spieler(NewesSpielDialog.Editfield2.text, rgb(255,255,0), 10, self)
Spieler(2) = New Spieler(NewesSpielDialog.Editfield3.text, rgb(0,255,0), 20, self)
Spieler(3) = New Spieler(NewesSpielDialog.Editfield4.text, rgb(255,0,0), 30, self)
```

Die Namen wollen wir im Spiel auch anzeigen, weshalb wir im Hauptfenster vier Static-Texte in die Nähe der Häuser setzen und sie „NameFeld“ nennen. Die Zeile: „NameFeld(i).Text = Spieler(i).Name“ tragen wir in eine der Schleifen (von 0 bis 3) in der Methode „NeuesSpiel“ ein. Dadurch füllen wir die Spielernamen mit dem Inhalt aus dem Dialog.

Spieler: Mensch oder Computer?

Beim Programmstart ist jeder Spieler zunächst automatisch ein Computerspieler. Mittels einer zusätzlichen Eigenschaft der Klasse Spieler mit dem Namen „Mensch“ und dem Typ „Boolean“ ändern wir dies. Wenn also ein Spieler ein Mensch ist, soll diese Eigenschaft den Wert True bekommen.

In der Methode „NeuesSpiel“ holen wir uns die Information, „Mensch oder Computer“ aus dem Dialog und halten sie in der Eigenschaft fest. Man kann die Wahl zwischen

Mensch und Computer im Dialog durch Checkboxes, Radiobuttons oder Pop-up-Menüs realisieren. Wir entscheiden uns hier für vier Pop-up-Menüs und benutzen zur Abfrage folgenden Code den wir ganz am Ende der Methode „NeuesSpiel“ eintippen:

```
Spieler(0).Mensch =
NeuesSpielDialog.PopupMenu1.IstIndex = 0
Spieler(1).Mensch =
NeuesSpielDialog.PopupMenu2.IstIndex = 0
Spieler(2).Mensch =
NeuesSpielDialog.PopupMenu3.IstIndex = 0
Spieler(3).Mensch =
NeuesSpielDialog.PopupMenu4.IstIndex = 0
```

Wählt man nun im Pop-up-Menü den Eintrag 0 für „Mensch“ aus, wird dieser Spieler zum Menschen erklärt.

Interaktive Spielzüge

Bis zu vier Menschen dürfen mitspielen. Dafür müssen wir kurz überlegen, wie das Mitspielen von statten geht: Der Computer würfelt für alle. Wenn man eine Sechs würfelt und eine Figur ins Spiel muss, dann braucht der Spieler nichts zu tun, der Rechner erledigt das für ihn. Lediglich die Figuren, die schon im Spiel sind, muss der menschliche Spieler „von Hand“ bewegen. Wird bei einem Spieler die Routine „GeheFelder“ aufgerufen, muss im Falle, dass dieser Spieler ein Mensch ist, das Spiel stoppen (Timer ausschalten).

Wir warten, bis der Spieler auf ein Feld vom Brett klickt. Hat dieses Feld eine Figur (Feld.Figur < > Nil) und gehört die Figur zum fraglichen Spieler (Feld.Figur.Spieler = Currentplayer), dann soll der Computer für den Spieler einen Zug mit der betreffenden Figur machen. Damit wir unnötige Arbeit vermeiden, teilen wir die GeheFelder-Methode auf und schaffen eine zweite Methode namens „GeheMitFigur“. Sie führt einen Zug für Computer oder Mensch mit einer vorgegebenen Figur durch. In GeheFelder rufen wir dann diese Methode für eine Computerfigur auf oder halten das Spiel für den Menschenzug an.

Die Umsetzung ist nicht besonders schwer, findet aber in dieser Ausgabe keinen Platz mehr. Auf der Abo-CD beziehungsweise im Internet finden Sie unseren Lösungsvorschlag. Damit läuft das Spiel schon vollständig und inklusive menschlichen Spielern.

Fazit

In dieser Folge haben wir gelernt, wie man menschliche Spieler in unsere Simulation einbaut. In der nächsten Ausgabe wollen wir uns mit der Vernetzung der Simulation befassen. Man kann unser Spiel dann mit mehreren Partnern an verschiedenen Orten übers Internet gleichzeitig spielen. *cm*

Tagesmenü In Realbasic konstruieren wir nicht nur Fenster, sondern auch das Programmmenü inklusive der Tastatur-Befehlskürzel (hier Befehl-Umschalt-N).

