Mac mogelt

Demonstration einer akustischen Täuschung auf dem Macintosh

Michael Bach, Computertechnik (c't) 1991, 5:274-282

Sinnestäuschungen sind vor allem für das visuelle System bekannt, es gibt aber auch akustische Täuschungen. Der vierstimmige Tongenerator des Macintosh ist gerade das Richtige für eine Live-Darbietung dieser Art. Unser Artikel beschreibt ein Pascal-Programm mit typischer Mac-Oberfläche, das nicht nur nett anzuhören, sondern auch noch recht lehrreich ist.

Unterthema: Listing Shepard_1

Anlaß für diese Veröffentlichung war eigentlich nur ein kleiner Disput mit der PC-Fraktion: Die behaupteten doch frech, eine halbwegs sinnvolle und nützliche Mac-Anwendung müsse bei all dem Bedienungs-Overhead und Fenster-Firlefanz zwangsläufig länger als 1000 Zeilen werden. Mein hier vorgestelltes Programm kommt, ohne auf Maus und Menüs verzichten zu wollen, auf nur 550 Zeilen - dafür ist sein Gebrauchswert aber auch recht beschränkt.

Immerhin erklärt es die recht komplizierte Programmierung der Mac-Events, der Tongeneratoren und der beliebten `Fahrstühle´ (Rollbalken) an einem nicht alltäglichen Beispiel: Es führt Ihnen nach erfolgreicher Kompilierung eine erstaunliche akustische Täuschung in Form der sogenannten Shepard-Töne zu Ohren.

Daneben wollte ich mich mit dem typischen Mac-Elementen auseinandersetzen: Handles, Menüs, Windows, Controls, Rollbalken und Dateizugriff. Deshalb sind in diesem Programm viele spielerische Elemente (z. B. eine effektvolle `About...´-Box) eingebaut, die für den Shepard-Effekt selbst nicht nötig wären. Die Informationen habe ich aus drei Quellen entnommen: den obligatorischen fünf Bänden `Inside Macintosh´ [2], dem Handbuch und den Beispielprogrammen von Turbo-Pascal und aus dem Think-Pascal-Handbuch (da steht aber nicht allzuviel drin). Empfehlenswert ist ferner der `Turbo Pascal Tutor Macintosh´ aus dem Hause Borland (nur in Englisch), der auch für THINK-Pascal-Anwender eine Fundgrube zur Mac-Programmierung ist. Leider spinnt der TP-Compiler unter Multifinder und MacroMaker; er wird von Borland zwar noch verkauft, aber nicht mehr aktualisiert, so daß er zur Programmentwicklung nur eingeschränkt empfohlen werden kann. Apples hauseigenes MPW dagegen ist ein Overkill für so ein kleines Projekt [4]. Apropos kleine Projekte: Wenn man beispielsweise vom PC-Turbo-Pascal gewöhnt ist, schnell mal ein kleines Programm einzutippen, und einem die Projekt-Verwaltung von THINK auf die Nerven geht, macht man sich ein für allemal ein Dummy-Projekt, in dem man immer ganz oben den neuen Code einfügt. Wird der mit einem `END.´ (der Punkt ist wichtig) abgeschlossen, kann der alte Kram hinten stehenbleiben und stört beim Übersetzen nicht im geringsten. Hat man sich erstmal an THINKs Eigenart der dauernden Quellcode-Formatierung gewöhnt, kann man sehr flott damit arbeiten: Ich tippe einfach fröhlich drauflos, ohne Tabs, Abstände und Zeilenenden, und der Editor macht immer `Pretty Print Pascal´ daraus.

Ohne Ressourcen

Entgegen `richtigen´ Mac-Programmen werden hier die Menüs, Fenster, Steuerelemente (Controls) und so

weiter nicht aus einer (mit ResEdit oder RMaker erzeugten) Ressource-Datei eingelesen, sondern im Programm selbst erzeugt. Deshalb ist es etwas schwieriger, das Programm zu `internationalisieren´, es ist aber konventioneller und demzufolge für Umsteiger durchsichtiger. Interessenten sei das Studium der Begleitschrift zu ResEdit empfohlen, die es bei Apples Entwicklerunterstützung zu kaufen gibt (kleiner Hinweis: lassen Sie sich nicht das alte ResEdit 1.2 andrehen. Version 2 ist der Hit!).

Getäuschte Sinne

Jeder Junginformatiker kennt D. Hofstadters Standardwerk `Gödel, Escher, Bach´ [3], das neben tiefschürfenden Weisheiten auch einige Illustrationen des niederländischen Grafikers Cornelius Escher enthält, zumeist optische Täuschungen und hirnverbiegende Darstellungen surrealer Räume und Gebäude. Eine davon - `Treppauf, treppab´ - ist das visuelle Analogon zu dem hier vorgestellten akustischen Effekt: Eine Anzahl Wächter läuft eine in sich geschlossene Burgtreppe hinauf, während andere dieselbe stetig hinabsteigen - ohne je oben oder unten anzukommen.

Der Shepard-Effekt

Durch eine trickreiche Manipulation von Tonhöhe und Oberwellenzusammensetzung entsteht der Eindruck eines unendlich ansteigenden Tones: Vier Töne, die um Oktavlage übereinanderliegen, werden 12mal um einen Halbton erhöht. Dabei wird die Lautstärke jedes der 4 Töne auf einer Skala von 0-24 wie folgt variiert: tiefster Ton von 0 auf 11, zweitunterster von 12 auf 24, zweitoberster von 24 auf 12 und oberster von 12 auf 0. Das bedeutet, daß man danach wieder mit dem ersten Ton anfangen kann, ohne daß ein Sprung auftritt. In Hofstadters schönem Buch ist dieser Effekt sehr anschaulich als Notensatz dargestellt. Der Ton wird laufend um einen Halbton höher, verschwindet jedoch nicht aus dem Hörbereich; darin besteht der paradoxe Wahrnehmungseindruck. Wenn man dies einmal gehört hat, reicht es fürs Leben, das ist mit vielen optischen Täuschungen auch so. Um sich das Prinzip der Täuschung zu verdeutlichen, kann man nach jedem Ton auf den Abwärtspfeil im Tonhöhen-Fahrstuhl klicken: Dann bleibt die Tonhöhe konstant, aber die Klangfarbe ändert sich.

Das Demonstrationsprogramm spart nicht am Bedienungskomfort und ist noch dazu MultiFinder-fähig

Klang-Computer

Von den verschiedenen Betriebsarten des Klangmoduls im Mac wird hier der 4stimmige Tongenerator benutzt. Man übergibt jeder Stimme eine `WAVE´, die hier aus einem Schwingungszug besteht. Nach verschiedenen Versuchen habe ich eine Dreiecksform gewählt. Zur individuellen Lautstärkeregelung der 4 Stimmen wird einfach die WAVE in unterschiedlicher Amplitude erzeugt. Um Knacksgeräusche zu reduzieren, wird die Lautstärke vor Verändern der Tonhöhe mit *decrescendo* heruntergeregelt, nachher mit *crescendo* wieder erhöht. Da die Berechnung der 24 WAVEs auf einem SE oder MacPlus langwierig ist (ca. 1/2 Minute, auf einem LC noch 8 Sekunden), werden die Daten als Datei abgespeichert. Ist die Datei schon vorhanden, entfällt die Berechnung. Ein Problem ist die Lautstärkeskala: Damit die 24 Lautstärkestufen gleichmäßig erscheinen, müssen sie logarithmisch und nicht etwa linear abgestuft sein; dies liegt an der Physiologie des Hörens. Nun ist dies mit 127 möglichen Lautstärkestufen (8-Bit-Auflösung) nur angenähert möglich. Die Konstante 1.26 (etwa die 24ste Wurzel aus 127) erwies sich als brauchbarer Faktor. Die Zahlen werden vom `Sound Driver´ als 8-Bit-Zweierkomplement interpretiert, das heißt, 0 entspricht -128, 128 entspricht 0 und 255 entspricht +127.

Die Tonhöhe wird für jede Stimme in der entsprechenden RATE übergeben. RATE ist eine `Fixed´-Zahl, die in `BuildNotes´ für eine wohltemperierte Stimmung berechnet wird (12*12te Wurzel aus 2 = 2 und damit 1 Oktave). Die `FixMath´-Unit muß nur wegen der Umrechnung der `Extended´-Gleitkommazahlen in `Fixed´ (die in ein LONGINT paßt) mit `x2fix´ eingebunden werden.

Wie im Inside Mac

Das Programm läuft nach den üblichen Initialisierungsorgien in der Standard-Ereignisschleife `repeat ... until doneFlag´, wie jedes anständige Mac-Programm. Zusätzliches Ereignis ist die Abfrage, ob ein `Shepard-Ton´ fertig ist und gegebenenfalls ein neuer zu erzeugen wäre. Das Ende kommt bei Klicken des `O. k.´-Knopfs oder wenn man `Quit´ aus dem `Ablage´-Menü wählt. Die Punkte im `Bearbeiten´-Menü sind nur aus Konsistenzgründen (und für evtl. aufgerufene DAs) drin und deshalb abgeschwächt dargestellt. Man sieht, was an Aufwand erforderlich ist, um so ein paar Kinkerlitzchen wie Fenster, Menüs und Rollbalken zu realisieren. Viele der kleinen Routinen, die den Programmausdruck so umfangreich machen, hat der erfahrene Mac-Programmierer in seinen Bibliotheken; in diesem `Steh-allein´-Programm müssen sie natürlich vollständig vorhanden sein. Dabei ist es noch erstaunlich einfach, die kompliziert anmutenden Rollbalken zu bedienen. In `HandleScrollBar´ muß nur entschieden werden, was beim Klicken auf Pfeile oder Fahrstuhl (`Thumb´) passieren soll, der Control Manager übernimmt den Rest. Die Shepard-Töne laufen immer, auch wenn man ein DA aufruft. Nur wenn einige der Routinen, die ich zum Testen brauchte, aus dem entsprechenden Menü gewählt werden, übernehmen sie den Tongenerator. (cm)

Literatur

[1] R. N. Shepard, Circularity in Judgements of Relative Pitch. Acoustic Society America 36:2346-2353 (1964)

[2] Apple Computer Inc, Inside Macintosh I-V, Addison-Wesley 1985-1990

[3] Douglas R. Hofstadter, Gödel, Escher, Bach, Klett-Cotta 1979

[4] Leo Drisis, Bequeme Übersetzer, Entwicklungsumgebungen für Pascal und C auf dem Macintosh, c't 8/90, S. 130-146

Kasten 1

Ein nicht unerheblicher Aufwand steckt in der korrekten Event-Behandlung, der Menü- und Fenster-Verwaltung. Routinen wie die Menüabfrage und die Event-Schleife gehören aber zum Standardrepertoire jedes Mac-Programmierers.

```
 0  program Shepard_1;
 1  { die akustische Täuschung eines scheinbar dauernd ansteigenden Tones }
 2  { 01/91 Dr. rer.nat. Michael Bach }
 3
 4    uses
 5         FixMath;
 6  { SANELib.lib & FixMath.p müssen mit ins Projekt! }
 7    const
 8         tMin = 0;     {niedrigster Ton}
 9         tMax = 84;   {höchster Ton}
10    lMin = 0;     {niedrigste Lautstärke}
11    lMax = 24;   {höchste Lautstärke}
12    menuCount = 4;
13    dName = 'ShepardWaves.DAT';
14    var
```

```
15    myMenus: array[1..menuCount] of MenuHandle;
16    myDuration, oldSoundLevel, mySoundLevel, myPitch, ShepIndex: integer;
17    myWaves: array[lMin..lMax] of Wave;
18    t: array[tMin..tMax] of longint;
19    tn: array[tMin..tMax] of string[4];
20    anFtSynth: FTSynthRec;
21    anFtSound: FTSoundRec;
22    EndTicks: longint;
23    EffektWindowPtr: WindowPtr;
24    myEvent: EventRecord;
25    OkControl, VolControl, SpeedControl, PitchControl: ControlHandle;
26    doneFlag: boolean;
27    dragRect: Rect;
28
29  { erst 'mal einige simple Routinen, aus unserer Bibliothek hierher kopiert }
30    function max (i1, i2: longint): longint;
31    begin
32    if i1 > i2 then
33      max := i1
34    else
35      max := i2
36    end;
37
38    function min (i1, i2: longint): longint;
39    begin
40    if i1 < i2 then
41      min := i1
42    else
43      min := i2
44    end;
45
46    function CenteredWindow (wWidth, wHeight: integer; title: Str255):
                               WindowPtr;
47  { Erzeuge Bildschirm-zentriertes Fenster }
48    var
49      windowBounds: Rect;
50    begin
51    with screenBits.bounds do
52      begin
53         windowBounds.left := ((right + left - wWidth) div 2);
54         windowBounds.top := ((bottom + top - wHeight) div 2);
55      end;
56    with windowBounds do
57      begin
58         bottom := top + wHeight;
59         right := left + wWidth;
60      end;
61    CenteredWindow := NewWindow(nil, windowBounds, title, true,
                        documentProc, Pointer(-1), false, 0);
62    end;
63
64    function MakeRect (xl, yt, xr, yb: integer): Rect;
65  { Erzeuge ein Rechteck (als Funktion) }
66    var
67      r: rect;
68    begin
69    SetRect(r, xl, yt, xr, yb);
70    MakeRect := r;
71    end;
72
73  {Breite eines Rechtecks}
74    function rctWidth (r: rect): integer;
75    begin
76    rctWidth := r.right - r.left;
```

```
 77    end;
 78
 79    procedure BuildNotes;
 80  { jetzt geht's richtig los, zunächst die Berechnung der Noten einer
       wohltemperierten Skala}
 81    const
 82      c = 1.059463094359295265;
 83  { =12. Wurzel aus 2 mit extended-Genauigkeit}
 84    var
 85      i, j: integer;
 86      d: extended;
 87      s: str255;
 88    begin
 89    d := 440 * c * c;
 90    s := 'C C#D D#E F F#G G#A BbB ';
 91    j := 0;
 92    for i := tMin to tMax do
 93      begin
 94         d := d * c;
 95         t[i] := x2fix(d * 44.93 * 256.0 * 0.0000001);   { Inside II-228 }
 96         tn[i] := concat(s[j + 1], s[j + 2], ' ');
 97         j := (j + 2) mod 24;
 98      end;
 99    end;
100
101    procedure BuildWave;
102  { Kurvenformen erzeugen, eine für jedes der Lautstärke-Niveaus  0-24 }
103    var
104      i, j: integer;
105      r1, r2: real;
106    begin
107    for i := 0 to 255 do
108      begin
109         r2 := sin(i / 255 * 2 * pi);
110         r1 := 127;
111         for j := lMax downto lMin do
112         begin
113           myWaves[j, i] := 128 + round(r2 * r1);
114           r1 := r1 / 1.26;
115  { ergibt die logarithmische Lautstärkeskala }
116         end;
117      end;
118    end; { Procedure BuildWave }
119
120    procedure GetWave;
121  { Kurvenform einlesen bzw. neu generieren }
122    var
123      f: file of wave;
124      aWindowPtr: windowPtr;
125      fndrInfo: FInfo;
126      i: integer;
127    begin
128    aWindowPtr := CenteredWindow(300, 100, ' Shepard-Töne ');
129    SetPort(aWindowPtr);
130    if GetFInfo(dName, 0, fndrInfo) = noErr then
131      begin {gibt's die Datei schon?}
132         MoveTo(50, 50);
133         DrawString('Kurvenform wird eingelesen');
134         open(f, dName);
135         for i := lMin to lMax do
136         read(f, myWaves[i]);
137         close(f);
138      end
139    else
```

```
140       begin
141          MoveTo(50, 50);
142          DrawString('Kurvenform wird neu berechnet');
143          BuildWave;
144          open(f, dName);
145          for i := lMin to lMax do
146          write(f, myWaves[i]);
147          close(f);
148       end;
149     DisposeWindow(aWindowPtr);
150     end;
151
152     procedure Crescendo;
153   { Ton sanft einschalten }
154     var
155       i, j: longint;
156     begin
157     for i := 0 to 7 do
158       begin
159          SetSoundVol((i * mySoundLevel) div 7);
160          if odd(i) then
161          delay(1, j);
162       end;
163     end;
164
165     procedure Decrescendo;
166   { Ton sanft ausschalten }
167     var
168       i, j: longint;
169     begin
170     for i := 6 downto 0 do
171       begin
172          SetSoundVol((i * mySoundLevel) div 6);
173          if odd(i) then
174          delay(1, j);
175       end;
176     end;
177
178     function MySoundDone: boolean;
179     begin
180     MySoundDone := TickCount >= EndTicks;
181     end;
182
183
184     procedure FourTone (f1: fixed; v1: integer; f2: fixed; v2: integer;
                          f3: fixed; v3: integer; f4: fixed; v4, d: integer);
185   { Ansteuerung der 4 Stimmen mit Frequenz fi & Lautstärke vi }
186     begin
187     Decrescendo;
188     anFTSynth.mode := ftmode;
189     anFTSynth.sndRec := @anFTSound;
190     with anFTSound do
191       begin
192          duration := d * 2;
193          sound1wave := @myWaves[v1];
194          sound2wave := @myWaves[v2];
195          sound3wave := @myWaves[v3];
196          sound4wave := @myWaves[v4];
197          sound1phase := 0;
198          sound2phase := 0;
199          sound3phase := 0;
200          sound4phase := 0;
201          sound1rate := f1;
202          sound2rate := f2;
```

```
203         sound3rate := f3;
204         sound4rate := f4;
205       end;
206     StartSound(@anFTSynth, sizeof(anFTSound) + sizeof(anFTSynth), nil);
207   { nil=synchron, dazu müssen 'anFTSynth' & 'anFTSound' global deklariert
      werden! }
208     Crescendo;
209     EndTicks := TickCount + d;
210     end; { Procedure UseFourTone }
211
212
213     procedure NotenTest;
214   { Hören wir uns `mal testweise kurz alle Töne an }
215     var
216       i: integer;
217     begin
218     for i := tMin to tMax do
219       FourTone(t[i], lMax, 0, 0, 0, 0, 0, 0, myDuration);
220     end;
221
222     procedure VolumeTest;
223   { Hören wir uns `mal testweise alle Lautstärkeniveaus an }
224     var
225       i: integer;
226     begin
227     for i := lMin to lMax do
228       begin
229          repeat
230          until MySoundDone;
231          FourTone(t[myPitch + 24], i, 0, 0, 0, 0, 0, 0, myDuration);
232       end;
233     end;
234
235     procedure AkkordTest;
236   { große Septime }
237     var
238       i: integer;
239     begin
240     for i := myPitch to myPitch + 12 do
241       begin
242          repeat
243          until MySoundDone;
244          FourTone(t[i], lMax, t[i + 4], lMax, t[i + 7], lMax, t[i + 11],
           lMax, myDuration * 2);
245       end;
246     end;
247
248     procedure Shepard;
249   { Berechnung der Frequenzen & Intensitäten für den Shepard-Effekt }
250     var
251       i: integer;
252     begin
253     ShepIndex := (ShepIndex + 1) mod 12;
254     i := ShepIndex;
255     TextFont(helvetica);
256     TextFace([bold]);
257     TextMode(srcCopy);
258     TextSize(24);
259     MoveTo(50, 100);
260     DrawString(tn[i]);
261     FourTone(t[myPitch + i], i, t[myPitch + 12 + i], 12 + i, t[myPitch +
24 + i], 23 - i, t[myPitch + 36 + i], 12 - i, myDuration);
262     end;
263
```

```
264    procedure UpdateControl (sc: ControlHandle);
265  { Texte mit neuen Werten an die Rollbalken, wenn sich Wert verändert hat }
266    var
267      s: str255;
268    begin
269    TextFont(SystemFont);
270    TextFace([]);
271    TextMode(srcCopy);
272    TextSize(12);
273    s := concat(' ', sc^^.contrlTitle, ' ', StringOf(sc^^.contrlValue), ' ');
274    MoveTo(sc^^.contrlRect.left + (rctWidth(sc^^.contrlRect) -
             StringWidth(s)) div 2, sc^^.contrlRect.top - 4);
275    DrawString(s);
276    end;
277
278    procedure HandleScrollBar (SC: ControlHandle; aPoint: point;
                                part: integer; var ResultVal: integer);
279  { Rollbalken (=Fahrstühle) behandeln }
280    var
281      StartUpTicks, DelayTicks: LongInt;
282    begin
283    HiliteControl(sc, part);
284    DelayTicks := round(150 / GetCtlMax(SC));
285  { gleich schnelle Bewegung für alle Rollbalken }
286    repeat
287      case part of
288         inUpButton:
289         SetCtlValue(SC, max(GetCtlMin(SC), pred(GetCtlValue(SC))));
290         inPageUp:
291         SetCtlValue(SC, GetCtlMin(SC));
292         inDownButton:
293         SetCtlValue(SC, min(GetCtlMax(SC), succ(GetCtlValue(SC))));
294         inPageDown:
295         SetCtlValue(SC, GetCtlMax(SC));
296         inThumb:
297         part := TrackControl(SC, aPoint, ptr(-1));
298      end;     { case }
299      Delay(DelayTicks, StartUpTicks);
300  { StartUpTicks ist nur ein Dummy }
301    until not StillDown;
302  { Bis Maustaste losgelassen, "Fahrstuhl" bewegen }
303    HiliteControl(sc, 0);
304    UpdateControl(sc);
305    ResultVal := GetCtlValue(SC);
306    end;
307
308    procedure DoAbout;
309  {und hier ein schickes «Über...« Fenster}
310    var
311      aWindowPtr: WindowPtr;
312      rct: Rect;
313      s: str255;
314      x, xs, winkel: integer;
315    begin
316    StopSound;
317    aWindowPtr := CenteredWindow(450, 150, ' Shepard-Töne ');
318    SetPort(aWindowPtr);
319    rct := aWindowPtr^.portRect;
320    InsetRect(rct, 8, 8);
321    s := 'Shepard RN (1964) Circularity in judgements of relative pitch.
           J Acoust  Soc Amer 36:2346-2353';
322    TextBox(@s[1], length(s), MakeRect(2, 2, 148, 148), teJustRight);
323    s := 'Realisiert von Michael Bach, Januar 1991. email: Bach@ruf.
           uni-freiburg. dbp.de';
```

```
324    TextBox(@s[1], length(s), MakeRect(302, 2, 448, 148), teJustLeft);
325    winkel := 0;
326    x := 148;
327    MoveTo(151, 1);
328    PenMode(PatXor);
329    xs := -1;
330    repeat
331      case winkel of
332         0:
333         Line(x, 0);
334         1:
335         Line(0, x);
336         2:
337         Line(-x, 0);
338         3:
339         Line(0, -x);
340      end;{case}
341      winkel := (winkel + 1) mod 4;
342      x := x + xs;
343      if abs(x) >= 148 then
344         begin
345         xs := -xs;
346         x := x + xs;
347         end;
348    until button;
349    DisposeWindow(aWindowPtr);
350    SetPort(EffektWindowPtr);
351    FlushEvents(everyEvent, 0);
352    end;
353
354    procedure DeleteMyFile;
355    var
356      anOSErr: OSErr;
357    begin
358    anOSErr := FSDelete(dName, 0);
359    end;
360
361    procedure UpdateMainWindow;
362  { alles neu malen, wenn Fensterinhalt
       beschädigt }
363    var
364      box: rect;
365    begin
366    SetPort(EffektWindowPtr);
367    DrawControls(EffektWindowPtr);
368    PenSize(3, 3);
369    box := OkControl^^.contrlRect;
370    InsetRect(box, -4, -4);
371    FrameRoundRect(box, 16, 16);
372    UpdateControl(VolControl);
373    UpdateControl(SpeedControl);
374    UpdateControl(PitchControl);
375    end;
376
377    procedure HandleMenu (mResult: longint);
378  { Menübefehle ausführen }
379    var
380      theItem, theMenu, i: integer;
381      name: str255;
382    begin
383    theItem := loWord(mResult);
384    theMenu := HiWord(mResult);
385    case theMenu of
386      1:     {Apfel-Menü}
```

```
387         if theItem = 1 then
388         doAbout
389         else
390         begin
391           GetItem(myMenus[1], theItem, name);
392           i := OpenDeskAcc(name);
393           SetPort(EffektWindowPtr);
394         end;
395       2:     {Bearbeiten}
396         case theItem of
397         1:
398           DeleteMyFile;
399         3:
400           doneFlag := true;
401         otherwise
402         end;
403       4:     {Tests}
404         case theItem of
405         1:
406           NotenTest;
407         2:
408           VolumeTest;
409         3:
410           AkkordTest;
411         end;{case}
412       otherwise
413     end;{case}
414     HiliteMenu(0);
415     end;
416
417     procedure HandleClick (theWindow: WindowPtr; where: point);
418   { reagiere auf Mausklick im Hauptfenster }
419     var
420       whichControl: ControlHandle;
421       i: integer;
422     begin
423     if theWindow <> FrontWindow then
424       SelectWindow(theWindow)
425     else
426       begin
427   { In welchem Control? }
428         GlobalToLocal(where);
429         i := FindControl(where, EffektWindowPtr, whichControl);
430         if whichControl = OkControl then
431         doneFlag := TrackControl(OkControl, where, nil) <> 0
432         else if whichControl = VolControl then
433         HandleScrollBar(VolControl, where, i, mySoundLevel)
434         else if whichControl = SpeedControl then
435         HandleScrollBar(SpeedControl, where, i, myDuration)
436         else if whichControl = PitchControl then
437         HandleScrollBar(PitchControl, where, i, myPitch);
438       end;
439     end;
440
441     procedure doMouseDown (theEvent: EventRecord);
442   { wo hat die Maus geklickt? }
443     var
444       whichWindow: WindowPtr;
445     begin
446     case FindWindow(theEvent.where, whichWindow) of
447       inMenuBar:
448         HandleMenu(MenuSelect(theEvent.where));
449       inContent:
450         HandleClick(whichWindow, theEvent.where);
```

```
451       inDrag:
452          DragWindow(whichWindow, theEvent.where, dragRect);
453       inSysWindow:
454          SystemClick(theEvent, whichWindow);
455       otherwise
456     end;{case}
457     end;
458
459     procedure HandleEvent (theEvent: EventRecord);
460   { was ist denn passiert? }
461     var
462       theChar: char;
463     begin
464     case theEvent.what of
465       mouseDown:
466          doMouseDown(theEvent);
467       keyDown, autoKey:
468          begin
469          theChar := chr(BitAnd(theEvent.message, charCodeMask));
470          if BitAnd(theEvent.modifiers, cmdKey) <> 0 then
471            HandleMenu(MenuKey(theChar))
472          else
473            doneFlag := theChar in [chr(13), chr(3)];
474          end;{keyDown}
475       updateEvt:
476          begin
477          BeginUpdate(WindowPtr(theEvent.message));
478          UpdateMainWindow;
479          EndUpdate(WindowPtr(theEvent.message));
480          end;{updateEvt}
481       otherwise
482     end;{case}
483     end;
484
485     procedure SetUpMenus;
486   { Menüleiste aufbauen, hier ausnahmsweise nicht aus Ressource heraus }
487     var
488       i: integer;
489     begin
490     myMenus[1] := NewMenu(1, chr(appleMark));
491     AppendMenu(myMenus[1], 'Über «Shepard»...;-');
492     AddResMenu(myMenus[1], 'DRVR');
493     myMenus[2] := NewMenu(2, 'Ablage');
494     AppendMenu(myMenus[2], 'Datei löschen;-;Quit/Q');
495     myMenus[3] := NewMenu(3, 'Bearbeiten');
496     AppendMenu(myMenus[3], '(Rückgängig/Z;-;(Ausschneiden/X;(Kopieren/C;
                              (Einfügen/V;(Löschen');
497     myMenus[4] := NewMenu(4, 'diverse Tests');
498     AppendMenu(myMenus[4], 'Alle Töne;Alle Lautstärken;Akkorde');
499     for i := 1 to menuCount do
500       InsertMenu(myMenus[i], 0);
501     DrawMenuBar;
502     end;
503
504     procedure Initialize;
505     begin
506   { Toolbox-Inits erledigt normalerweise THINK Pascal, hier zur
        Verdeutlichung: }
507     InitGraf(@thePort);
508     InitFonts;
509     FlushEvents(everyEvent, 0);
510     InitWindows;
511     InitMenus;
512     TEInit;
```

```
513    InitDialogs(nil);
514    MaxApplZone;
515    InitCursor;
516    GetWave;
517    BuildNotes;
518    GetSoundVol(OldSoundLevel);
519    mySoundLevel := 7;
520    myPitch := 20;
521    myDuration := 20;
522    EndTicks := TickCount;
523    ShepIndex := 0;
524    SetUpMenus;
525  { Bewegungsraum für Fenster auf dem Bildschirm }
526    SetRect(dragRect, 4, 24, screenBits.bounds.right - 4,
              screenBits.bounds.bottom - 4);
527    EffektWindowPtr := CenteredWindow(400, 200, ' Shepard-Töne ');
528    SetPort(EffektWindowPtr);
529  { Controls einsetzen, hier OK-Button und drei Rollbalken }
530    OkControl := NewControl(EffektWindowPtr, MakeRect(330, 160, 390,
                    160 + 20), 'Ok', true, 0, 0, 0, pushButProc, 0);
531    VolControl := NewControl(EffektWindowPtr, MakeRect(150, 25, 300,
         25 + 16), 'Lautstärke', true, mySoundLevel, 0, 7, scrollBarProc, 0);
532    SpeedControl := NewControl(EffektWindowPtr, MakeRect(150, 90, 300,
         90 + 16), 'Tondauer', true, myDuration, 5, 50, scrollBarProc, 0);
533    PitchControl := NewControl(EffektWindowPtr, MakeRect(150, 160, 300,
         160 + 16), 'Tonhöhe', true, myPitch, tMin, tMax - (11 + 3 * 12),
         scrollBarProc, 0);
534    UpdateMainWindow;
535    doneFlag := false;
536    end;
537
538    procedure Cleanup;
539  { wir räumen ordentlich auf }
540    begin
541    DisposeWindow(EffektWindowPtr);
542    Decrescendo;
543    StopSound;
544    SetSoundVol(OldSoundLevel);
545    end;
546
547  { Initialisierung machen wir selbst - also: }
548  {$I-}
549  begin
550  { Event-Hauptschleife }
551    Initialize;
552    repeat
553    SystemTask;   {für den Multifinder und DAs}
554    if MySoundDone then     {neuer Ton fällig?}
555      Shepard;
556    if GetNextEvent(everyEvent, myEvent) then
557      HandleEvent(myEvent);
558    until doneFlag;
559    Cleanup;
560  end.
```

Zu diesem Artikel existieren Programmbeispiele

0591_274.sea