STRUKTURIERTES PROGRAMMIEREN

Strukturiertes Programmieren ist keineswegs so schwierig wie es sich anhört. Neben etwas Theorie wollen wir ein kleines aber ausbaufähiges Spiel entwerfen.

Im letzten Heft gab ich Ihnen einige Regeln, die das Strukturieren des Programmcodes betreffen. Eigentlich war dieser Bericht zu früh dran. Denn bevor man sich an seinen Computer setzt und das Programm eintippt, ist eine Menge Vorarbeit zu leisten. Die wichtigste Arbeit besteht dabei im Nachdenken. Nachdenken vor allem darüber, was das Programm, das man schreiben will, eigentlich tun soll, wie man es am effektivsten aufbaut. Und dabei werden die meisten und verhängnisvollsten Fehler gemacht. Diese Fehler ziehen sich durch das gesamte »Software-Projekt« hindurch und entscheiden später über den Erfolg oder Mißerfolg des Programms. Und je später ein Fehler endeckt wird, desto größer wird der Aufwand, den man betreiben muß, um diesen Fehler zu beheben. Das bedeutet ganz einfach, daß man am Anfang sehr aufmerksam sein muß, und jeden Schritt, jede Idee lieber einmal mehr als einmal zu wenig überdenken sollte.

Gerade dann, wenn das Programm wahrscheinlich etwas größer wird, tut man gut daran, es in kleinere Einheiten aufzuteilen. Diese Einheiten werden auch Module genannt. Gebräuchlich sind auch die Begriffe »Unterprogramm«, »Subroutine« oder auch »Prozeduren« (zum Beispiel in Pascal). Gemeint ist in der Regel immer das gleiche.

Bei großen und komplexen Programmpaketen besteht eine der wichtigsten Aufgaben in der Modularisierung eines Programms. Das heißt aber auch, daß, wenn diese Aufgabe erste einmal gelöst ist, der Rest um so einfacher und schneller von der Hand geht.

Wie ein Spiel entsteht

Wenden wir uns jetzt einem neuen Beispiel zu. Ich möchte mit Ihnen ein kleines und einfaches Spiel entwerfen. Die Spielregeln sind sehr einfach. Wir wollen den Computer veranlassen, gegen uns zu spielen. In ein Spielfeld von 3 x 3 Feldern setzen die Spieler (der Computer und wir) abwechselnd eine Null oder ein X. Wir setzen die X. Wer als erster drei Zeichen in einer Reihe hat, hat gewonnen. Eine Reihe heißt hier senkrecht, waagerecht oder diagonal. Das Spielfeld und eine typische Spielsituation sind in Bild 1a und 1b zu sehen.

x	0	
x		x
0		

1	2	3
4	5	6
7	8	9

Bild 1a, b, c. Das Spielfeld wird eine typische Spielsituation

Wir wollen unser Spielfeld jetzt numerieren, jedes Feld erhält eine Nummer (Bild 1c).

Jetzt schreiben wir uns auf, was in jedem Feld steht:

```
1 x   2 0   3 —
4 x   5 —   6 x
7 0   8 —   9 —
```

Ein Strich bedeutet hier ein leeres Feld.

```
Ersetze  x durch 1
         0 durch —1
         — durch 0
```

Der Grund dafür, daß wir gerade diese Zahlen nehmen, ist jetzt noch nicht so klar, im Moment ist es einfach ein Gefühl für Symmetrie. Wir werden aber sehen, daß diese Wahl zweckmäßig ist.

Damit können wir den Spielstand so beschreiben:

Feld	Inhalt
1	1
2	—1
3	0
4	1
5	0
6	1
7	—1
8	0
9	0

Array SS (Spielstand)

Diesen (und auch jeden anderen) Spielstand legen wir in einem Array SS ab (SS steht für Spielstand). So oft sich also der Spielstand ändert, ändert sich auch der Inhalt des Arrays SS. SS(5) enthält also immer den Inhalt des mittleren Feldes, eine 1, wenn wir einen Stein (x) auf das Feld gesetzt haben, eine —1, wenn der Computer seinen Spielstein darauf gesetzt hat, oder eine 0, wenn das Feld noch nicht besetzt ist.

Jetzt können wir (und vor allem der Computer) herausfinden, welches Feld besetzt ist, und auch, durch wen es besetzt wurde, aber wir müssen den Compuer ja noch dazu bringen, einen vernünftigen Spielzug zu machen. Das bedeutet, er muß sperren, wenn wir schon zwei Felder einer Reihe besetzt haben, und er soll auch erkennen, wenn er schon zwei Felder einer Reihe besetzt hat, wo er seinen Siegzug hinzusetzen hat. Wir brauchen also eine Methode, um den Spielstand zu bewerten.

Dazu schaffen wir ein zweites Array, das wir BW nennen (BW steht für Bewertung). BW enthält die Summen jeder Reihe, der waagerechten, der senkrechten und der diagonalen. Insgesamt haben wir acht Reihen, so daß auch BW nicht größer zu sein braucht. Und so wird jedes Element von BW berechnet:

```
1    0 = SS(1)+SS(2)+SS(3)   oberste Reihe
2    2 = SS(4)+SS(5)+SS(6)   zweite Reihe
3   —1 = SS(7)+SS(8)+SS(9)   dritte Reihe
4    1 = SS(1)+SS(4)+SS(7)   linke Spalte
5   —1 = SS(2)+SS(5)+SS(8)   mittlere Spalte
6    1 = SS(3)+SS(6)+SS(9)   rechte Spalte
7    1 = SS(1)+SS(5)+SS(9)   vordere Diagonale
8   —1 = SS(3)+SS(5)+SS(7)   hintere Diagonale
```

Diese Werte geben selbstverständlich nur den oben im Beispiel vorgegebenen Spielstand wieder. Aber wir können jetzt erkennen, daß das Array BW uns Informationen über die aktuelle Spielsituation liefert: Immer, wenn ein Element von BW den Wert 2 hat, erkennt der Computer, daß für ihn Gefahr im Verzuge ist. Die 2 bedeutet nämlich, daß in einer Reihe schon zweimal ein Sein gesetzt wurde. Damit dürfte die nächste Aufgabe des Computers schon klar sein: Er muß seinen nächsten Stein in das dritte Feld der fast kompletten Reihe setzen! Doch wie soll er das freie Feld finden? Wir machen es uns einfach: