Fachhochschule für Wirtschaft Berlin (FHW)

Lehrveranstaltung: Objektorientierte Programmierung 1
(LV 315201.02)

Aufgabe 4 : Magische Quadrate

Dozent: Prof. Lang

Vorgelegt von: Elenio Mattera

Berlin, den 20.01.2009

Inhaltsverzeichnis

Was ist ein Magisches Quadrat?

Ein **magisches Quadrat** mit der Kantenlänge *n* wird durch die Anordnung der Zahlen *1,2...n^2* unter Berücksichtigung bestimmter Kriterien gebildet.

So muss bei einem klassischen „magischen Quadrat“ jede **Zeilen**-, **Spalten-** und **Diagonalsumme** einen bestimmten Wert haben, damit man dieses als „magisch“ bezeichnen kann.

Die s.g. „**magische Zahl**“ wird durch die Formel *i = ((Seitenlänge^3) + Seitenlänge) / 2* berechnet.

Nehmen wir den Kasus Seitenlänge(*n*) = 4, so ist die magische Zahl = ((4^3)+4)/2, also **34**.

Problem

Die Probleme sind ganz offensichtlich:

1) Jede der Zahlen von 1 – 16 darf nur einmalig verwendet werden.
2) Die Summen jeder Zeile, Spalte und den beiden Diagonalen muss 34 ergeben.

Aufgabe

Die Aufgabe bestand nun darin, diese beiden Probleme zu bewältigen, und möglichst viele magisches Quadrate - unter der Verwendung von drei Initialwerten - zu erzeugen.

Der „Rohling“ sieht also dementsprechend aus:

	12		
	15	14	

Unter diesen Voraussetzungen gibt es natürlich nur eine limitierte Anzahl an Lösungen.

Außerdem erübrigt sich durch die Initialwerte nun ein Lösungsversuch mit „System“, wie z.B. die „**Big X“** Methode, bei der man von seiner „aktuellen Zelle“ in einem bestimmten Schema die umliegenden Zellen beschreiben muss.

Lösung

Der Endalgorithmus muss also flexibel auf die Initialwerte reagieren und trotzdem zum Ziel kommen. Ich habe mich in ein meiner Belegarbeit für eine rekursive Backtrackinglösung entschieden, bei der zuerst alle Zellen befüllt und anschließend versucht wird, die aktuelle Zelle soweit zu erhöhen, dass die Regeln zur Lösung des magischen Quadrats eingehalten werden. Da er an Position 16 schon den maximalen Wert eingetragen hat, kann er diese Zelle nicht mehr erhöhen; also setzt er diese auf „0“ und springt in die davor liegende Zelle und

erhöht diese, springt wieder zurück in die Zelle von welcher er „gekommen“ ist (also wieder die 16.) und setzt dort den Wert von der davor liegenden Zelle ein. Verallgemeinert kann man sagen, dass er die Nachbarzahlen solange vertauschen, bis das Quadrat magisch ist.

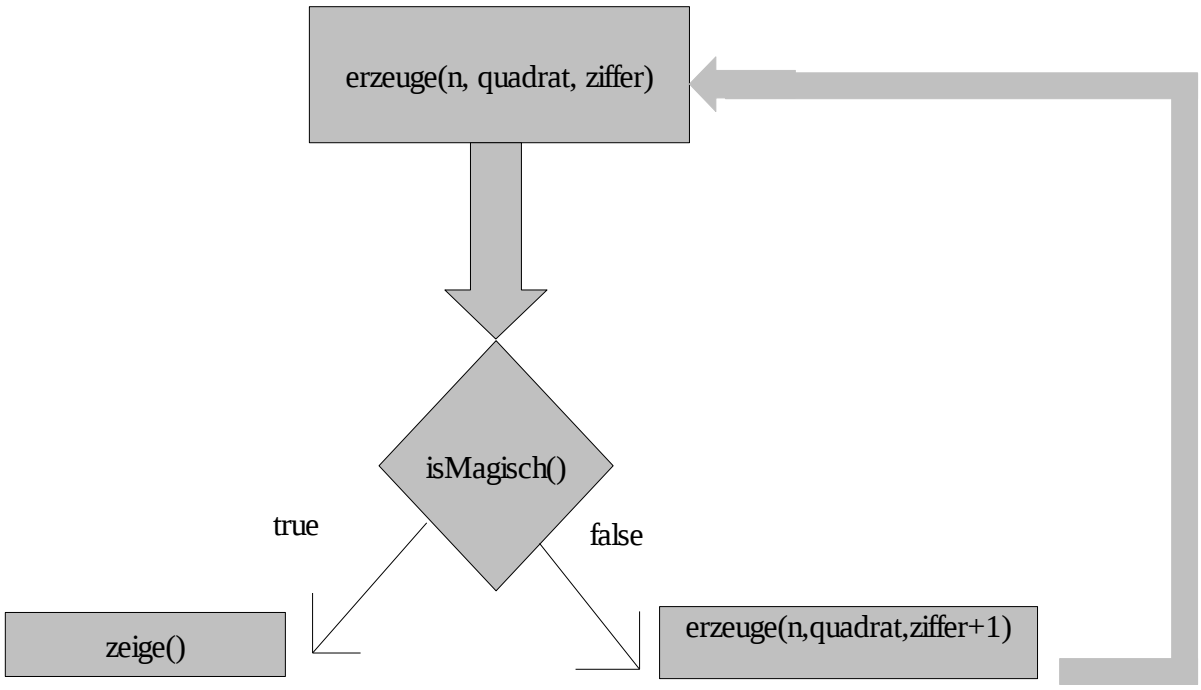
Nach 13 Schritten ist das Programm also an dieser Position:

Schritt 14	**Schritt 16**	**Schritt 18**	**Schritt 19**
1 2 3 4 5 12 6 7 8 9 10 11 13 15 14 16	1 2 3 4 5 12 6 7 8 9 10 11 16 15 14 13	1 2 3 4 5 12 6 7 8 9 10 13 11 15 14 0	1 2 3 4 5 12 6 7 8 9 10 13 11 15 14 16

Wie man sieht lässt das Programm die Initialwerte unberührt und vertauscht immer den nächst kleineren Wert, der nicht als Initialwert deklariert wurde. Den Zwischenschritt mit den setzen der „0“ ist bei Schritt 19 erkennbar.

Ich habe mein Programm so aufgebaut, dass für jeden Schritt eine eigene Methode vorgesehen wurde.

Die **Main()-Methode** führt das Programm ganz klassisch aus und führt praktisch die anderen Methoden zusammen. Bei der **initialisieren()-Methode** wird das Quadrat mit seinen Vorgabewerten befüllt. Außerdem schreibt sie in alle leeren Zellen „0“. Die **erzeuge()-Methode** ist das Herz meines Programms, hier werden die einzelnen Quadrate erzeugt, indem sich die Methode immer wieder selbst mit einem anderen Parameter aufruft.



Die **erzeuge()-Methode** wird durch die Methoden **checkZiffer()** und **isMagisch()** geleitet.

Am Methodenanfang wird geprüft, ob das Quadrat bereits magisch ist: wenn ja, beende den Aufruf und gib das Ergebnis aus. Falls die Antwort false ist, soll er ganz normal weiter machen.

Die **checkZiffer()** Methode überprüft jede Zahl einzeln auf ihre Einzigartigkeit. So wird der Arbeitsschritt direkt unterbrochen, wenn dem nicht so ist und die **erzeuge()-Methode** wird mit einem anderen Parameter wieder neu aufgerufen.

(Optional kann auch die etwas statischere Methode erzeugeSchnell() verwendet werden. Da diese auf die checkZiffer() - Funktion verzichtet und die Initialwerte manuell implementiert wurden ist diese um einiges schneller; ca. 5min – im Vergleich zu 20min)

Zu guter Letzt wird das Gesamtergebnis durch die **zeigen()-Methode** ausgegeben. Hierbei arbeiten sich 2 for-Schleifen Zelle für Zelle durch das Array und geben jeden Wert einzeln aus.

Weitere Informationen zu den Methoden sind via JavaDoc in Projekt selbst zu finden.

Mögliche Modifikationen

Da die aktuelle Version meiner Lösung nur ein magisches Quadrat ausgibt, bestünde natürlich noch Handlungsbedarf, um die Aufgabe als „voll erfüllt“ anzusehen.

Um mehrere magische Quadrate für diese Konstellation zu generieren, müssten man eine Art Zwischenspeicher, in Form eines Containers erstellen, welche die richtigen 2D-Arrays speichert.

Des weiteren müsste man die verbesserte **erzeuge()** Funktion dazu bewegen können, nach einem gefunden Quadrat eine weitere Zelle mit irgendeinem Wert zu fixieren, sodass der 2. Lösungsversuch unweigerlich anders als der 1. sein muss.