Programmieren mit dem ClassPad II

Dr. Wolfgang Ludwicki
Dr. Jens Weitendorf

1. Auflage - Oktober 2014

Inhaltsverzeichnis

Einleitung ... 1
1. Strukturiertes Programmieren und Rekursion mit dem ClassPad II ... 3
1.1 Sequenz ... 4
1.2 Einseitige Auswahl ... 6
1.3 Zweiseitige Auswahl ... 8
1.4 Wiederholung mit vorangestelltem Test (kopfgesteuerte Schleife) ... 9
1.5 Gezählte Wiederholungen ... 11
1.6 Wiederholung mit nachgestelltem Test (fußgesteuerte Schleife) ... 15
2. Parameter in Programmen ... 17
2.1 Konfigurieren der Parametervariablen und Eingabe ihrer Werte ... 17
2.2 Lokale Variablen ... 18
3 Unterprogramme ... 19
4 Rekursion ... 23
Beispiel 1 für Rekursion ... 24
Beispiel 2 für Rekursion ... 25
Beispiel 3 für Rekursion ... 25
Beispiel 4 für Rekursion ... 26
Beispiel 5 für Rekursion ... 26
Beispiel 6 für Rekursion ... 27
Beispiel 7 für Rekursion ... 29
Beispiel 8 für Rekursion ... 30
5. Programme für den Mathematikunterricht ... 33
5.1 Würfeln mit dem ClassPad II ... 33
Erzeugung einer Verteilung ... 33
Das „Raab“-Problem ... 36
Das Problem der vollständigen Serie ... 37
5.2 Wachstumsprozesse ... 39
Lineares Wachstum ... 39
Exponentielles Wachstum ... 40
Beschränktes Wachstum ... 41
Das logistische Wachstum ... 42
5.3 Mathematische Experimente ... 44

Grafische Spielereien ... 44
Spiel mit dem Zufall und Termverständnis ... 45
Das Newton-Verfahren für komplexe Funktionen ... 50
Das Apfelmännchen ... 52
Der Random-Walk ... 53
Der zweidimensionale Fall ... 55
5.4 Programme zur Simulation stochastischer Prozesse ... 56
Das Münzenproblem ... 56
Das Ziegenproblem ... 57
Das Schlüsselproblem ... 58
Das Problem Traummann ... 60
Die Fußball Bundesliga ... 63
6 Literatur und Quellen ... 64

Einleitung

In den 70er und 80er Jahren des letzten Jahrhunderts blieb einem in der Regel nichts anderes übrig, als zu programmieren, wenn man den Computer mathematisch nutzen wollte. Dies hat sich im Laufe der Zeit grundlegend geändert, da inzwischen eine große Anzahl mathematischer Software entwickelt worden ist, die dafür verantwortlich ist, dass das Programmieren in der Schule ein Schattendasein führt; dies gilt teilweise auch für den Informatikunterricht. Dabei sprechen die beiden folgenden Argumente unbedingt dafür, dass im Mathematikunterricht Programme eingesetzt werden.

Wenn man ein Programm erstellt, setzt dies eine Durchdringung des Problems voraus; das kann dazu führen, dass schon das Erarbeiten eines Programms das Problem löst, so dass man es gar nicht mehr laufen lassen muss. Des Weiteren kann man Programme dazu nutzen, mathematische Aussagen zu testen oder Hypothesen aufzustellen. Eine der im Unterricht zu fördernden mathematischen Kompetenzen ist: „Mit Mathematik symbolisch / formal / technisch umgehen“. So trägt das Programmieren zur Klarheit hinsichtlich des Variablen-, Term- und Funktionsbegriffs bei. Insbesondere werden auch die verschiedenen Aspekte des Gleichheitszeichens geklärt.

Die Anwendung neuer Technologien hat inzwischen auch in die mathematische Forschung Einzug gehalten.

Papier und Bleistift wurden lange als die einzigen Hilfsmittel der Mathematiker betrachtet (einige würden hier auch noch den Papierkorb dazuzählen). Wie in vielen anderen Gebieten auch, hat allerdings in den letzten Jahren der Einsatz von Computern in der Mathematik stark zugenommen und somit die Rolle des Experiments in der Mathematik erweitert und legitimiert. Wie können Mathematiker den Computer als Werkzeug benutzen? Und kann er vielleicht nicht nur als Werkzeug, sondern sogar als Mitarbeiter eingesetzt werden? [1]

Für Schülerinnen und Schüler ist eine formale mathematische Argumentation in der Regel nicht überzeugend. In diesem Zusammenhang können durch den Rechner gewonnene Ergebnisse den Wahrheitsgehalt einer Aussage stützen. HESSE nennt dieses abduktives Begründen. (s. [2], S. 126)

Natürlich setzt der Einsatz von Programmen voraus, dass die Schülerinnen und Schüler die Fähigkeit des Programmierens erworben haben. Bezogen auf die mathematischen Programme, die im zweiten Teil beschrieben werden, sind nur geringe Programmierfähigkeiten erforderlich. Für diejenigen, die tiefer in das Programmieren einsteigen wollen, ist der erste Teil gedacht.

So wird zunächst das strukturierte Programmieren an Beispielen aus der Mathematik erläutert, wobei auch auf die Verwendung rekursiver Unterprogramme eingegangen wird. Im umfangreichen Abschnitt 5 werden Programme für den Mathematikunterricht diskutiert, die die oben genannten Aspekte beispielhaft verdeutlichen. Einige der

diskutierten Probleme lassen sich auch mit Hilfe der Tabellenkalkulation behandeln. Die Benutzung des Programm-Moduls bietet in der Regel die Möglichkeit, den Stichprobenumfang hinreichend groß zu wählen.

Alle in diesem Text angegebenen Programme stehen auf

www.casio-schulrechner.de/de/lehrerschule/materialdatenbank/

zum Download bereit.

1. Strukturiertes Programmieren und Rekursion mit dem ClassPad II

Das strukturierte Programmieren ist gekennzeichnet durch die baumartige Zerlegung eines Programmes in Teilprogramme (Module, Unterprogramme, Prozeduren) und die Beschränkung auf die drei Kontrollstrukturen (Steueranweisungen, Algorithmenbausteine)

- Sequenz (lineare Folge von Anweisungen),
- Selektion (Auswahl, Programmverzweigung),
- Repetition (Wiederholung, Schleife, Loop).

Entscheidenden Einfluss auf das Entstehen des strukturierten Programmierens hatten EDSGER W. DIJKSTRA und NIKLAUS WIRTH.

Mit der Programmiersprache Pascal hatte N. WIRTH ein Werkzeug geschaffen, das sich in hervorragender Weise zum Erlernen des strukturierten Programmierens eignet.

Von ISSAC NASSI und BEN SHNEIDERMAN wurde eine grafische Form der Darstellung für Algorithmen entwickelt, die die Umsetzung des Entwurfes des Algorithmus in ein strukturiertes Programm fast automatisiert. Die Nassi-Shneiderman-Diagramme, deren Grundform Rechtecke sind, werden auch Struktogramme genannt. (vgl. [9])

Mit dem Einsatz von CAS-Rechnern im Mathematikunterricht gewinnt das algorithmische Denken an Bedeutung, indem Aufgaben in Teilaufgaben zerlegt werden, deren Lösungen zur Gesamtlösung zusammengesetzt werden. Die Lösungen der Teilaufgaben können als Ergebnis der Anwendung von Unterprogrammen betrachtet werden.

Die Programmiersprache des ClassPad II ähnelt stark einer BASIC-Sprache. BASIC-Sprachen unterstützen von sich aus das strukturierte Programmieren nicht. In der Programmiersprache des ClassPad II sind jedoch alle Steuerelemente enthalten, mit denen Selektion und Repetition realisiert werden können. Einschränkungen für das strukturierte Programmieren beim ClassPad II ergeben sich durch Mängel bei der Variablenübergabe bei Unterprogrammen (Subroutines).

Beim ClassPad II gibt es auch den Sprungbefehl goto, der einen unbedingten Sprung zu einer durch ein Label gekennzeichneten Stelle erlaubt. Beim strukturierten Programmieren wird jedoch auf die Verwendung dieses Sprungbefehles verzichtet. Die Vermeidung des goto-Befehls ist ein wesentliches Merkmal des strukturierten Programmierens.

In den ersten Abschnitten wird gezeigt, wie Struktogramme mithilfe der Programmiersprache des ClassPad II realisiert werden können. Damit wird auch dargestellt, wie sich die Lernenden das Programmieren mit dem ClassPad II aneignen können. Weiterhin wird an Beispielen demonstriert, dass mit der Programmiersprache des ClassPad II rekursive Problemlösungen möglich sind.

Der Umgang mit dem Programmteil des ClassPad II wird hier nicht im Detail erläutert. Das ist ausführlich in [4], [5] und [6] geschehen. Es wird hier vorausgesetzt, dass das Editieren, Öffnen, Speichern und Ausführen von Programmen auf dem ClassPad II bekannt ist.

Alle folgenden Programme wurden mithilfe eines *ClassPad Manger v3 Professional Version 3.06.0000.2240* hergestellt und getestet. Die Programme wurden im Ordner *main* gespeichert.

Die Bildschirmfotos (Screenshots) wurden mithilfe eines *ClassPad Manager for ClassPad II Series Version 01.00.0000.2300 for Windows* hergestellt.

1.1 Sequenz

Eine lineare Folge von Anweisungen wird auf dem ClassPad II realisiert, indem jede Anweisung in eine neue Zeile geschrieben wird oder die Anweisungen durch einen Doppelpunkt „:“ getrennt werden.

Alle Eingaben erfolgen im Menü „Programm“. Da ein neues Programm eingegeben werden soll, wird aus dem Menü „Edit“ „Neue Datei“ ausgewählt. Der Typ ist „Progr. (Normal)“, als Ordner wird „main“ eingestellt, als Name wird eine Zeichenkette eingegeben.

Beispiel 1:

Es ist der Flächeninhalt eines Dreiecks zu berechnen, von dem die drei Seitenlängen bekannt sind. Für die Lösung bietet sich die Heronische Dreiecksformel an. Folgendes Programm löst diese Aufgabe. Es handelt sich dabei um eine Folge (Sequenz) von Anweisungen.

Der Befehl „ClrText“ wird aus dem Menü „I/O“ - „Freigeben“ ausgewählt. Dieser Befehl löscht den Text im Programmausgabefenster.

Zum Befehl „Input“, der sich im Menü „I/O“ - „Eingabe“ befindet, ist im Handbuch zu lesen:

> „„Syntax: Input G <Variablenname>[,"<Zeichenkette 1>"[,"<Zeichenkette 2>"]]
>
> Funktion: Wenn die Programmausführung den Input-Befehl erreicht, wird der Anwender aufgefordert, einen Wert einzugeben, der dann der angegebenen Variablen zugeordnet wird.
>
> Beschreibung:
>
> - Wenn Sie für "<Zeichenkette 1>" nichts angeben, wird standardmäßig die Eingabeaufforderung „<Variablenname>?“ angezeigt.
> - Der für "<Zeichenkette 2>" angegebene Text wird als Titel für das Eingabedialogfeld verwendet.

- Dieser Befehl unterbricht die Programmausführung und zeigt ein Dialogfeld an, das die durch "<Zeichenkette 1>" angegebene Textzeichenkette und ein Eingabefeld enthält. Für "<Zeichenkette 1"> kann eine in Anführungszeichen (" ") eingeschlossene Zeichenkette oder ein Variablenname angegeben werden.
- Wenn eine lange Textzeichenkette eingegeben wird, kann es sein, dass diese bei der Anzeige im Dialogfeld abgeschnitten wird.
- Wenn das Dialogfeld angezeigt wird, geben Sie einen Wert in das Eingabefeld ein, und tippen Sie dann auf [OK]. Dadurch wird das Dialogfeld geschlossen, der eingegebene Wert wird der entsprechenden Variablen zugeordnet, und die Programmausführung wird wiederaufgenommen.
- Wenn Sie im Dialogfeld auf [Cancel] tippen, wird die Programmausführung beendet.

Hinweis:

- Während der Ausführung des Input-Befehls wird die Programmausführung für die Dateneingabe unterbrochen. Während das Programm unterbrochen ist, können Sie nur einzelne mathematische Ausdrücke eingeben. Sie können keine Befehle oder eine Serie von durch Doppelpunkte (:) getrennten Anweisungen eingeben.“ (Handbuch, Seite 220)

Die Wertzuordnung für die Variablen erfolgt mittels ⇒. Dieser Befehl befindet sich im Menü „Strg“. Das Handbuch bemerkt zu diesem Befehl:

„Syntax 1: {<Ausdruck> ; "<Zeichenkette>"} ⇒ <Variablenname>

Syntax 2: {<Ausdruck> ; "<Zeichenkette>"} ⇒ <Listenelement>

Syntax 3: <Ausdruck> ⇒ <Matrixelement>

Funktion: Der Inhalt des Ausdrucks auf der linken Seite wird ausgewertet, und das Ergebnis wird dem Element auf der rechten Seite zugeordnet.“ (Handbuch, Seite 213)

Zur Ausgabe wird der Befehl „Print“ verwendet, der sich im Menü „I/O“ - „Ausgabe“ befindet. Im Handbuch ist dieser Befehl so erläutert:

„Syntax 1: Print G <Ausdruck>[,<Farbbefehl>]

Syntax 2: Print G "<Zeichenkette>"

Funktion: Zeigt das Ergebnis des angegebenen Ausdrucks oder die angegebene Textzeichenkette an.

Beschreibung: Das Ergebnis eines Ausdrucks wird als einzelne Zeile angezeigt. Wenn das Ergebnis ein langer Ausdruck, ein Bruch oder eine Zeichenkette ist, passt es möglicherweise nicht auf das Display. Verwenden Sie in diesem Fall stattdessen den PrintNatural-Befehl.“ (Handbuch, Seite 224)

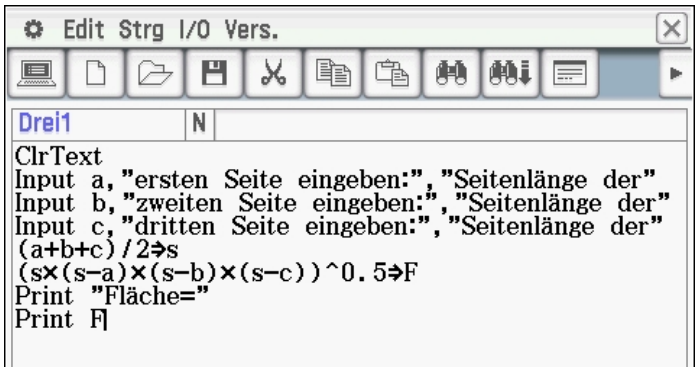


Abb. 1 Dreiecksfläche mit heronischer Formel

Das ist die Umsetzung des folgenden Struktogramms.

DREIECKSFLAECHE
Eingabe: a
Eingabe: b
Eingabe: c
(a+b+c)/2=>s
(s*(s-a)*(s-b)*(s-c))^0.5=>F
Ausgabe: "Fläche="
Ausgabe: F

```
ClrText
Input a,"ersten Seite
eingeben:","Seitenlänge der"
Input b,"zweiten Seite eingeben:",
"Seitenlänge der"
Input c,"dritten Seite eingeben:",
"Seitenlänge der"

(a+b+c)/2⇒s

(s*(s-a)*(s-b)*(s-c))^0.5⇒F

Print "Fläche="

Print F
```

1.2 Einseitige Auswahl

Beispiel 2:

Vier Zahlen werden eingegeben und mit der kleinsten beginnend der Größe nach sortiert ausgegeben. Es wird die Idee des Bubblesort benutzt. Der Tausch erfolgt mit der Hilfsvariablen *h*. Als Kontrollstruktur wird die einseitige Auswahl verwendet.

Die einseitige Auswahl wird mit der Kontrollstruktur „**If - Then - IfEnd**“ realisiert. Die entsprechenden Befehlswörter befinden sich im Menü „Strg“ - „If“.

Im Handbuch wird dazu erläutert:

„Syntax 1: If G <Ausdruck> : Then : [<Anweisung>] ... : IfEnd

Funktion 1:

- Wenn der Ausdruck TRUE (Wahr) ist, wird die Anweisung im Then-Block ausgeführt. Danach fährt die Ausführung mit der auf IfEnd folgenden Anweisung fort.
- Wenn der Ausdruck FALSE (Falsch) ist, fährt die Programmausführung mit der auf IfEnd folgenden Anweisung fort, ohne dass die Anweisung im Then-Block ausgeführt wird.“ (Handbuch, Seite 219)

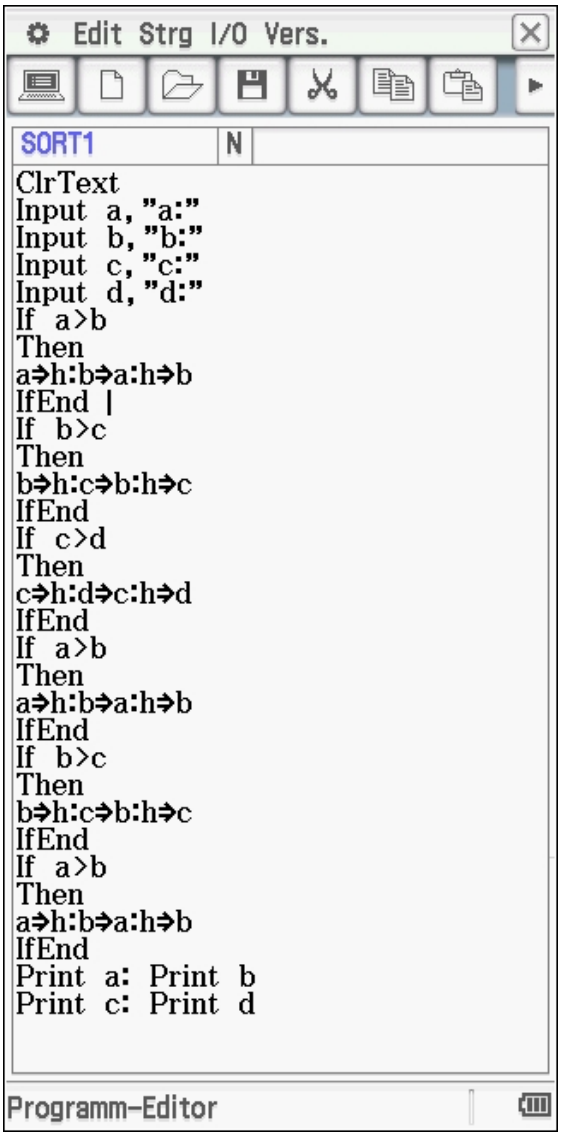
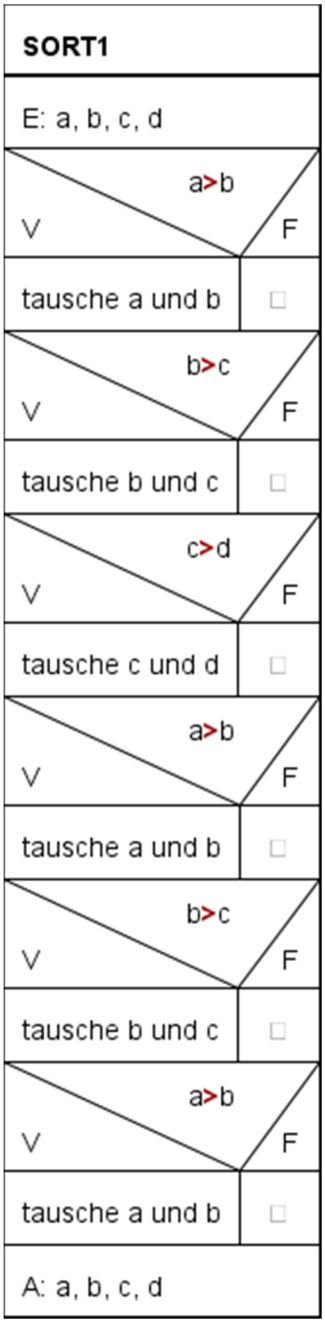


Abb. 2 Sortieren von vier Zahlen




```
ClrText
Input a,"a:"
Input b,"b"
Input c,"c"
Input d,"d"
If a>b
Then
a⇒h : b⇒a : h⇒b
IfEnd
If b>c
Then
b⇒h : c⇒b : h⇒c
IfEnd
If c>d
Then
c⇒h : d⇒c : h⇒d
IfEnd
If a>b
Then
a⇒h : b⇒a : h⇒b
IfEnd
If b>c
Then
b⇒h: c⇒b :h⇒c
IfEnd
If a>b
Then
a⇒h : b⇒a : h⇒b
IfEnd
Print a : Print b
Print c : Print d
```

1.3 Zweiseitige Auswahl

Beispiel 3:

Von einem Dreieck mit den Seitenlängen a, b und c ist festzustellen, ob es rechtwinklig ist.

Der Satz des Pythagoras darf als notwendige und hinreichende Bedingung für die Rechtwinkligkeit benutzt werden.

Bei dem angegebenen Algorithmus wird nicht überprüft, ob es überhaupt ein Dreieck gibt, dessen Seitenlängen die eingegebenen Zahlen sind. Die Einbeziehung dieser Untersuchung ist eine sinnvolle Erweiterung und Übungsaufgabe.

Die zweiseitige Auswahl wird durch die Kontrollstruktur „If - Then - Else -IfEnd“ verwirklicht. Das Handbuch führt dazu aus:

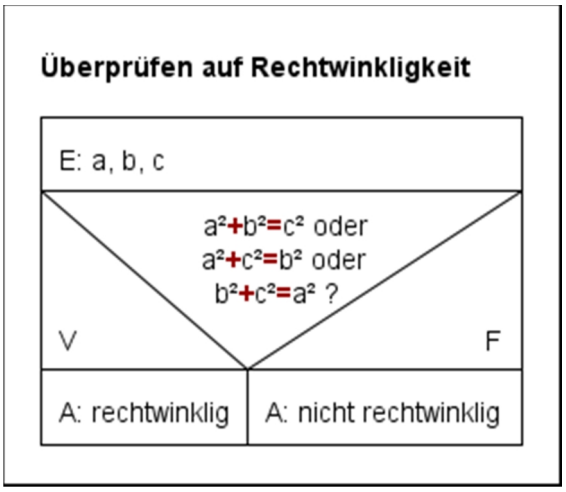
„Syntax 2:

If G <Ausdruck> : Then : [<Anweisung>] ... : Else : [<Anweisung>] ... : IfEnd

Funktion 2:

- Wenn der Ausdruck TRUE (Wahr) ist, wird die Anweisung im Then-Block ausgeführt. Danach fährt die Ausführung mit der auf IfEnd folgenden Anweisung fort.
- Wenn der Ausdruck FALSE (Falsch) ist, wird der Ausdruck im Else-Block anstelle des Then-Blocks ausgeführt. Danach fährt die Ausführung mit der auf IfEnd folgenden Anweisung fort.“ (Handbuch, Seite 219)

Es ist zu beachten, dass „Then“, „Else“ und „IfEnd“ Befehle sind. Deshalb müssen sie durch Wagenrücklauf oder „:“ vom vorausgehendem und nachfolgendem Befehl getrennt werden.



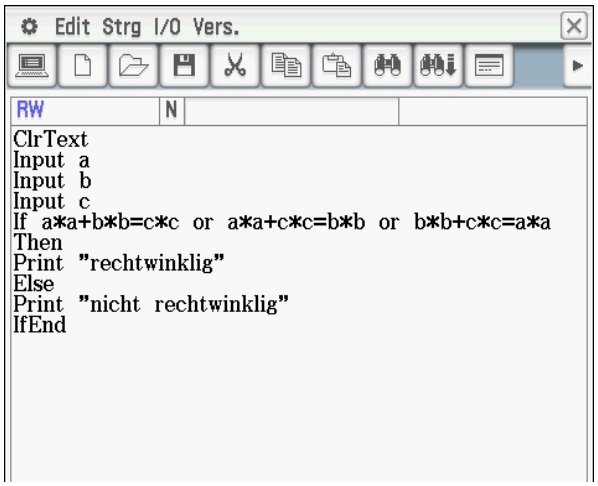
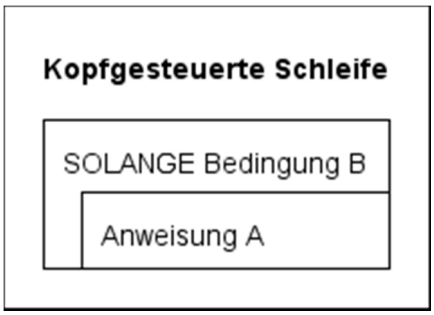


Abb. 3 Dreieck auf Rechtwinkligkeit prüfen

1.4 Wiederholung mit vorangestelltem Test (kopfgesteuerte Schleife)

Das Struktogramm für die Wiederholung mit vorangestelltem Test hat folgende Form.



Mit ClassPad II-Befehlen wird diese Schleife durch

While *B*

Anweisung A

WhileEnd

realisiert.

Solange die Bedingung B erfüllt ist, wird der Schleifenkörper (Anweisung A) wiederholt. Wenn die Bedingung B nicht erfüllt ist, wird der Schleifenkörper übersprungen und die nächste Anweisung ausgeführt. Da die Erfüllung der Bedingung B vor dem Ausführen des Schleifenkörpers geprüft wird, kann es passieren, dass der Schleifenkörper gar nicht ausgeführt wird, wenn nämlich bereits am Anfang die Bedingung nicht erfüllt ist. Daher auch die Bezeichnung abweisende Schleife.

Die Befehlswörter „While“ und „WhileEnd“ befinden sich im Menü „Strg“ - „While“. Die Kontrollstruktur „While - WhileEnd“ wird im Handbuch auf Seite 233 erläutert.

Beispiel 4:

Die Wahrscheinlichkeit dafür, dass ein Neugeborenes ein Knabe ist, beträgt etwa $p = 51,4\%$.

Wie viele Kinder n muss eine Familie mindestens haben, wenn mit einer Wahrscheinlichkeit von mindestens $f = 90\%$ mindestens 2 Mädchen darunter sein sollen? Das ist eine typische Dreimal-Mindestens-Aufgabe für die Binomialverteilung. Wenn X die Anzahl der Knaben in einer Familie bezeichnet, dann ist die Anzahl n der Kinder gesucht, so dass $P(X \le n-1) = B_{n;p}(\{0,1,2,\cdots,n-2\}) \ge f$.

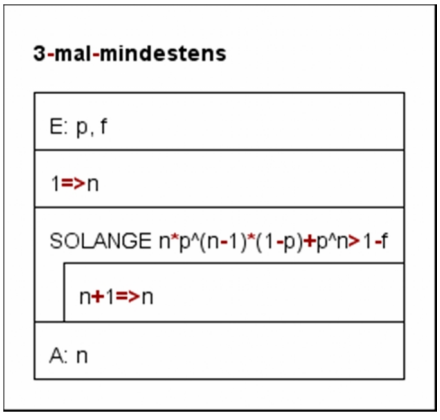
Folgende Umformungen führen zur Ermittlung von n:

$$B_{n;p}(\{0,1,2,\cdots,n-2\}) = 1 - B_{n;p}(\{n-1,n\}) \ge f,$$

$$B_{n;p}(\{n-1,n\}) \le 1-f,$$

$$n \cdot p^{n-1} \cdot (1-p) + p^n \le 1-f.$$

Der folgende Algorithmus, der eine kopfgesteuerte Schleife verwendet, löst diese sogenannte dreimal-mindestens-Aufgabe.



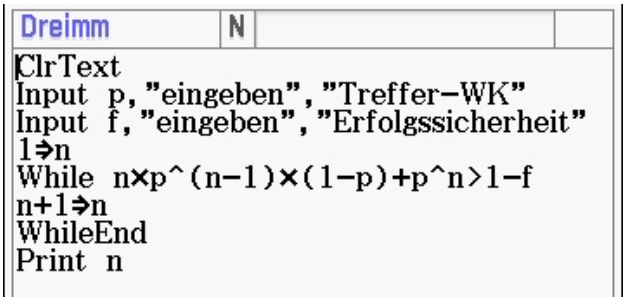


Abb. 4 Dreimal-Mindestens-Aufgabe

Eine Familie muss mindestens 7 Kinder haben, wenn mit mindestens 90%-iger Sicherheit mindestens zwei Mädchen dabei sein sollen.

1.5 Gezählte Wiederholungen

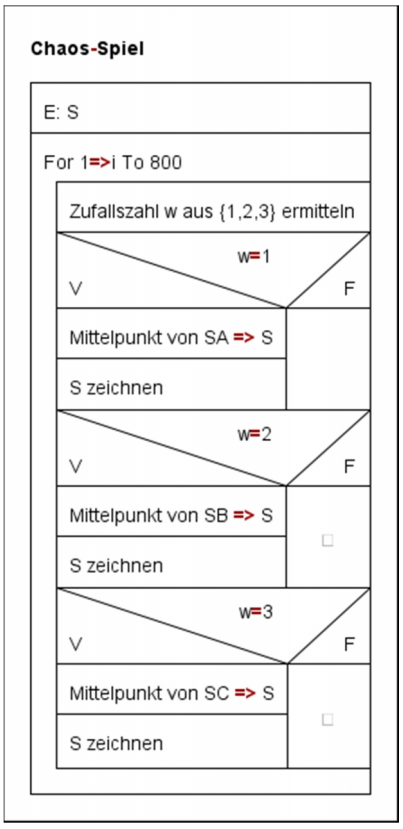
Beispiel 5:

Mithilfe des Chaosspiels, das von M. BARNSLEY gefunden wurde, kann ein Muster erzeugt werden, dass sich einem Sierpinski-Dreieck annähert. (vgl. [12], S. 353ff.)

In einer Ebene seien A, B und C die Eckpunkte eines Dreiecks. Die Zufallsgröße W nimmt gleich verteilt die Werte 1, 2 oder 3 an. Am Anfang wird zufällig ein Spielpunkt S gewählt und ein Wert von W ermittelt. Das kann mit einem Würfel geschehen, indem die Ereignisse „Augenzahl 1 oder 6“, „Augenzahl 2 oder 5“ und „Augenzahl 3 oder 4“ als 1, 2 bzw. 3 gedeutet werden. Nimmt W den Wert 1 an, so ist der nächste Spielpunkt der Mittelpunkt der Strecke SA. Nimmt W den Wert 2 an, so ist der nächste Spielpunkt der Mittelpunkt der Strecke SB. Nimmt W den Wert 3 an, so ist der nächste Spielpunkt der Mittelpunkt der Strecke SC. Von dem neuen Spielpunkt ausgehend, wird der nächste Spielpunkt nach derselben Vorschrift ermittelt. Die Spielpunkte werden gezeichnet. Mit zunehmender Spieldauer nähert sich das Muster einem Sierpinski-Dreieck.

In diesem Beispiel ist das Dreieck ABC gleichschenklig mit $A(0/0)$, $B(4/0)$ und $C(2/3)$.

Für den ersten Spielpunkt kann ein beliebiges Paar von Zahlen eingegeben werden.




```
ChaosSp  N
ClrGraph
Input sx,"eingeben", "x-Koordinate von S"
Input sy,"eingeben", "y-Koordinate von S"
For 1⇒i To 800
rand(1,3)⇒w
If w=1
Then
sx/2⇒sx : sy/2⇒sy : PlotOn sx,sy,ColorBlack
IfEnd
If w=2
Then
(4+sx)/2⇒sx : sy/2⇒sy : PlotOn sx,sy,ColorBlack
IfEnd
If w=3
Then
(2+sx)/2⇒sx : (3+sy)/2⇒sy : PlotOn sx,sy,Color
Black
IfEnd
Next
```

Abb. 5 Chaosspiel

Abb. 6 Sierpinski-Dreieck

Die gezählte Wiederholung wird mit der Kontrollstruktur „**For - To - Next**“ realisiert. Diese Kontrollstruktur kann durch die Schrittweite „Step“ erweitert werden. Die entsprechenden Befehlswörter sind im Menü „Strg“ - „For“ zu finden. Im Handbuch wird erklärt:

„Syntax:

For G <Ausdruck 1>⇒<Steuervariablenname> G To G <Ausdruck 2> [Step G <Ausdruck 3>] [<Anweisung>] ... : Next

- <Ausdruck 1> ist der Startwert, <Ausdruck 2> ist der Endwert und <Ausdruck 3> ist die Schrittweite.

Funktion: Alles, was zwischen dem For-Befehl und dem Next-Befehl steht, wird basierend auf einer Zählung wiederholt, die beim Startwert der Steuervariablen beginnt und endet, wenn die Steuervariable ihren Endwert erreicht. Bei jedem Durchlauf wird die Steuervariable um den durch den Step-Wert angegebenen Wert geändert. Die Schleife wird auch dann beendet, wenn der Wert der Steuervariablen den Endwert überschreitet.

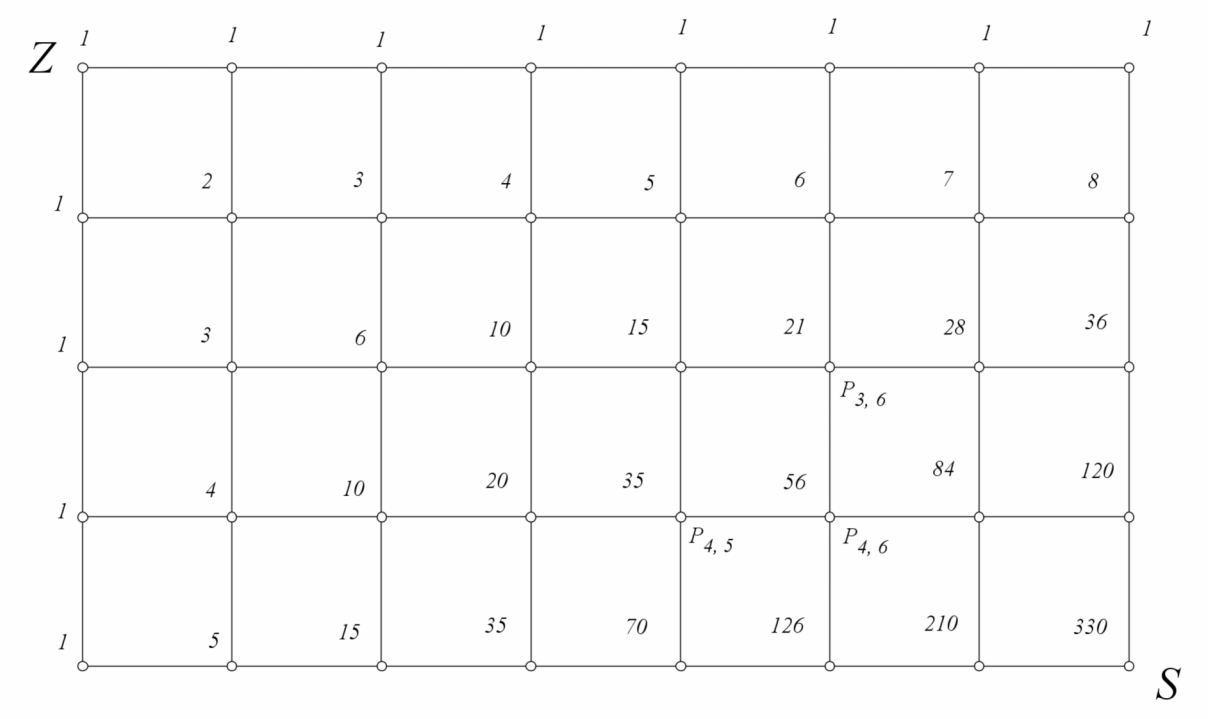
Beschreibung:

- Wenn kein Step-Wert angegeben wird, wird als Schrittweite 1 verwendet.
- Der Startwert kann kleiner als der Endwert sein, sofern als Schrittweite ein positiver Wert angegeben wird. In diesem Fall wird der Wert der Steuervariablen bei jedem Durchlauf um die Schrittweite erhöht.
- Der Startwert kann größer als der Endwert sein, sofern als Schrittweite ein negativer Wert angegeben wird. In diesem Fall wird der Wert der Steuervariablen bei jedem Durchlauf um die Schrittweite verringert.
- Sie können einen Mehrfachanweisungsbefehl (:) anstelle eines Wagenrücklaufs verwenden, um Anweisungen voneinander zu trennen.
- Verwenden Sie nicht den Goto-Befehl, um eine For~Next-Schleife zu verlassen.“ (Handbuch, Seite 218)

Beispiel 6:

In diesem Beispiel treten gezählte Wiederholungen geschachtelt auf.

In einem Stadtteil gibt es w waagerechte Straßen, die zueinander parallel sind, und s Straßen, die zu den waagerechten Straßen senkrecht verlaufen. Auf wie viel verschiedenen Wegen kann man, ohne Umwege zu machen, von einer der vier äußeren Ecken des Stadtteils zu der diagonal gegenüberliegenden Ecke gelangen?



Die Abbildung zeigt einen Stadtteil mit 5 waagerechten Straßen und 8 senkrechten Straßen. Gesucht ist hier die Anzahl der kürzesten Wege von S nach Z.

Bei Z beginnend werden die senkrechten Straßen mit $s_1, s_2, s_3, \cdots, s_s$ und die waagerechten Straßen mit $w_1, w_2, w_3, \cdots, w_w$ bezeichnet. Mit $P_{i,j}$ wird die Kreuzung bezeichnet, die von den Straßen w_i und s_j gebildet wird. Der Startpunkt S entspricht demnach der Kreuzung $P_{w,s}$.

Mit $wege(m,n)$ wird die Anzahl der verschiedenen Wege bezeichnet, auf denen man, ohne Umwege zu machen, von der Kreuzung $P_{m,n}$ zur Ecke Z gelangen kann.

Die Lösung der Aufgabe liefert schließlich $wege(w,s)$.

Die Funktion $wege(m,n)$ kann rekursiv definiert werden:

$$wege(m,n)=\begin{cases}1 & \text{für } m=1 \text{ oder } n=1\\ wege(m-1,n)+wege(m,n-1) & \text{sonst}\end{cases}$$

Eine direkte Umsetzung dieses rekursiven Ansatzes ist auf dem ClassPad II nicht möglich, weil im Definiens anwenderdefinierter Funktionen nur elementare arithmetische Ausdrücke vorkommen dürfen.

Zur Lösung der Aufgabe wird iterativ eine Matrix erzeugt, wie sie für den Fall $w=5$ und $s=8$ in der Abbildung angedeutet ist.

Der ClassPad-Befehl fill(1,w,s) erzeugt eine Matrix, deren Elemente alle den Wert 1 haben und die w Zeilen und s Spalten hat (vergleiche Handbuch Seite 77). Diese Matrix erhält den Namen $wege$.

Wege1 | N

```
ClrText
Input w,"Anzahl der waagerechten Wege"
Input s,"Anzahl der senkrechten Wege"
fill(1,w,s)⇒wege
For 2⇒i To w
For 2⇒j To s
wege[i-1,j]+wege[i,j-1]⇒wege[i,j]
Next
Next
Print "Anzahl der Wege:"
Print wege[w,s]
```

Abb. 7 Anzahl der kürzesten Wege in einem Stadtteil

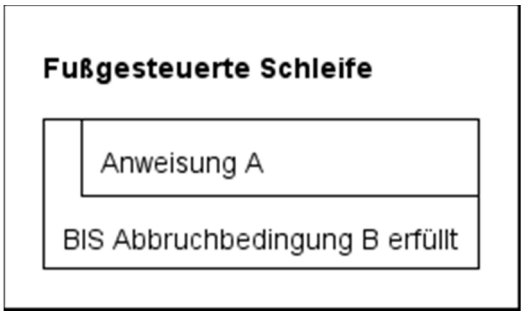
In [13] wird auf Seite 27 die Anzahl der kürzesten Wege kombinatorisch ermittelt:

„Wie viele kürzeste Gitterwege gibt es von (0,0) nach (m, n) mit $m,n\geq 0$?

Einem solchen Weg entspricht eineindeutig eine Folge von $m+n$ Zeichen ↑ (für „hoch“) und → (für „rechts“), wobei genau m-mal → und n-mal ↑ gewählt werden muss. Legen wir in der Reihe die Plätze für die ↑'s fest, so sind die der →'s auch festgelegt (die übrigen nämlich). Für diese Wahl hat man genau $\binom{m+n}{n}=\binom{m+n}{m}$ Möglichkeiten.“

1.6 Wiederholung mit nachgestelltem Test (fußgesteuerte Schleife)

Das Struktogramm für fußgesteuerte Schleifen hat die folgende Form. Im Vergleich dazu ist die Realisierung der fußgesteuerten Schleife mit ClassPad-Befehlen angegeben.




```
Do
  Anweisung A
LpWhile not(B)
```

Der Schleifenkörper *Anweisung A* wird wiederholt, wenn die Bedingung, die hinter **LpWhile** steht, wahr (TRUE) ist. Wenn diese Bedingung nicht erfüllt ist, wird die nächste Anweisung außerhalb der Schleife ausgeführt (vergleiche Handbuch Seite 218). Da erst jeweils nach dem Abarbeiten des Schleifenkörpers überprüft wird, ob die Wiederholungsbedingung erfüllt ist, wird der Schleifenkörper mindestens einmal ausgeführt. Deshalb auch die Bezeichnung nichtabweisende Schleife.

In den Struktogrammen wird bei diesen Schleifen eine Abbruchbedingung verwendet. Ist die Bedingung *B* die Abbruchbedingung, so stellt **not(B)** die Wiederholungsbedingung dar.

Beispiel 7:

Wenn *n* einstellig ist, dann ist die Spiegelzahl von *n* die Zahl *n* selbst. Wenn die Zahl *n* mehrstellig ist, dann ergibt sich die Spiegelzahl von *n*, indem die Ziffern von *n* in umgekehrter Reihenfolge aufgeschrieben werden. So ist die Spiegelzahl von 6 die Zahl 6 selbst, die Spiegelzahl von 6135 ist die Zahl 5316.

Der folgende Algorithmus SZ1 bestimmt zu einer natürlichen Zahl *n* deren Spiegelzahl.

SZ1 | N

```
ClrText
Input n,"natürliche Zahl eingeben:"
Print n
0⇒ergeb
Do
int(n/10)⇒rest: 'Zahl ohne Einer
mod(n,10)⇒ziffer: 'Einerziffer
10×ergeb+ziffer⇒ergeb: 'nächste Ziffer anhängen
rest⇒n: 'neues n
LpWhile not(rest≤0)
Print ergeb
```

Abb. 8 Erzeugen der Spiegelzahl einer natürlichen Zahl

Ergänzend wird noch ein Algorithmus SZ2 zur Ermittlung der Spiegelzahl angegeben, der eine gezählte Schleife auf der Grundlage der Stellenanzahl int(log(n)) der natürlichen Zahl *n* verwendet.

SZ2 | N

```
ClrText
Input n, "Natürliche Zahl eingeben:"
Print n
int(log(n))⇒st: 'um 1 verringerte Stellenanzahl von
 n
n⇒a
0⇒sz
For 0⇒i To st
mod(a,10)⇒b: 'Einerziffer von a
int(a/10)⇒a: 'a ohne Einerziffer
sz+b×10^(st-i)⇒sz: 'Aufbau der Spiegelzahl
Next
Print sz
```

Abb. 9 Erzeugen der Spiegelzahl mit Hilfe der Stellenanzahl

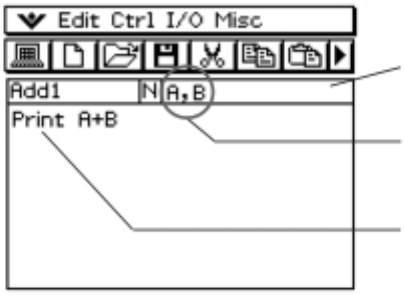
2. Parameter in Programmen

2.1 Konfigurieren der Parametervariablen und Eingabe ihrer Werte

Zu diesem Thema sei hier das Handbuch Seite 12-2-7 zitiert:

Sie können während der Programm-Erstellung oder beim Editieren im Programmeditor-Fenster Variablennamen in das Eingabefeld für Parametervariable eingeben, die das Programm während seiner Ausführung benutzen soll. Damit sind Sie dann in der Lage, beim Programmstart in das Programmlader-Fenster Parameterwerte einzugeben.

Beispiel

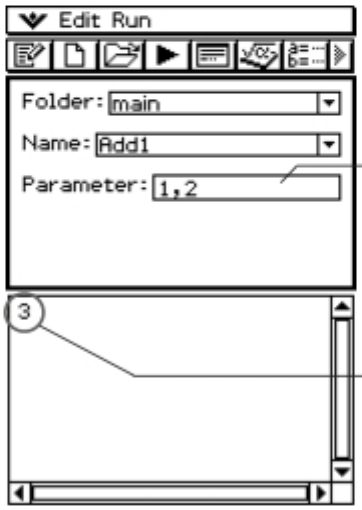


Programmeingabe

Eingabefeld für Parametervariable

A, B zeigt an, dass diese Variablen innerhalb des Programms verwendet werden.

Während der Programmausführung wird nach Eingabe von Werten für „A“ und „B“ die Summe der zwei Variablen berechnet und ausgegeben.



Programmlader-Fenster

Eingabefeld für die Parameterwerte
Geben Sie 1, 2 ein, bevor Sie die Programmausführung beginnen. Das Programm rechnet dann mit A=1 und B=2.

Durch die Programmausführung wird das Ergebnis A+B = 1+2 = 3 berechnet und angezeigt.

Tipp

- Falls Sie ein Programm ablaufen lassen wollen, das Parametervariable enthält, geben Sie unbedingt korrekte Werte für die Parameter ein. Es kommt auch zu einem Fehler, wenn die Anzahl der von Ihnen eingegebenen Werte nicht mit der Anzahl der Parametervariablen übereinstimmt.

2.2 Lokale Variablen

Das Handbuch Seite 12-2-8 bemerkt zu dieser Thematik:

Eine lokale Variable ist eine Variable, die temporär erstellt und in einem Programm verwendet werden kann. Verwenden Sie den **Local**-Befehl, um eine lokale Variable zu erstellen.

Syntax: Local□<Variablenname> (□ bezeichnet ein Leerzeichen)

Beispiel: Local abc Dieses Beispiel erstellt eine mit „abc" benannte lokale Variable.

Tipp

- Lokale Variablen werden automatisch gelöscht, nachdem die Ausführung des Programms beendet ist.
- Beachten Sie, dass die lokalen Variablen in ihrem eigenen, speziellen Ordner abgespeichert werden, sodass die lokalen Variablennamen die Namen anderer Variablen im Speicher des ClassPad nicht beeinträchtigen. Deshalb können Sie unbesorgt lokale Variablennamen im Programm erneut verwenden, selbst wenn dieser Variablenname bereits für einen anderen Variablen-Typ außerhalb dieses Programms benutzt wird.
- Variablen, die als Parametervariablen in einem Programm benutzt werden, werden automatisch als lokale Variablen behandelt. Die mit dem **Define**-Befehl erzeugten Variablen werden ebenfalls automatisch als lokale Variablen behandelt.

3 Unterprogramme

Die Verwendung von Unterprogrammen ist ein Wesensmerkmal des strukturierten Programmierens. Erst die Verwendung von Unterprogrammen ermöglicht die modulare Bearbeitung von Programmieraufgaben.

Ein Unterprogramm ist eine Folge von Anweisungen, die selbst ein Programm darstellen kann. Diese Folge von Anweisungen erhält einen Namen. Im aufrufenden Programm wird durch die Namensnennung des Unterprogramms, die Ausführung der Anweisungen des Unterprogramms veranlasst. Damit sind für den Umgang mit einem Unterprogramm Vereinbarung und Aufruf des Unterprogramms wichtig.

In höheren Programmiersprachen ist mit den Unterprogrammen häufig eine ausgeklügelte Parameterübergabe verbunden.

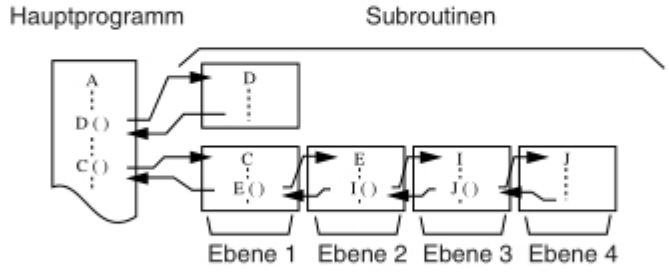
Durch das Verwenden von Unterprogrammen wird einerseits die Übersichtlichkeit von Algorithmen erhöht, andererseits werden die Programme kürzer, weil Codewiederholungen vermieden werden können.

Zu Unterprogrammen kann man im Handbuch Seite 12-2-8 lesen:

Durch das Einschließen einer anderen Programmdatei in ein Programm springt die Ausführung in die vorgegebene Programmdatei. Das den Sprung ausführende Programm wird als das „Hauptprogramm" bezeichnet, hingegen das Programm, an das gesprungen wird, als eine „Subroutine" bezeichnet wird.
Wenn die Programmausführung in das Hauptprogramm zurückkehrt, dann setzt sie an dem Punkt unmittelbar nach dem Sprungbefehl in die Subroutine fort.

Tipp

- Beachten Sie, dass jedes Programm als Subroutine verwendet werden kann. Gekennzeichnet ist eine Subroutine dadurch, dass von einem anderen Programm dorthin gesprungen wird.



Subroutinen können auf verschiedene Weise verwendet werden, um die Berechnungen einfacher zu gestalten. Wollen wir annehmen, dass Sie eine Formel haben, die mehr als einmal in einem Programm ausgeführt werden muss, oder die durch eine Anzahl unterschiedlicher Programme berechnet werden soll. Speichern Sie einfach diese Formel als separate Programmdatei (Subroutine) ab. Danach können Sie bei Bedarf immer diese Programmdatei, welche die Formel enthält, als Subroutine aufrufen.

Weiter werden im Handbuch Seite 12-2-9 folgende Beispiele angegeben:

Beispiel 1: Springen in eine Subroutine ohne Zuordnung von Werten zu den Parametervariablen der Subroutine

Hauptprogramm
Input A
Input B
Sub1( ) ← Springt zur Subroutine „Sub1"
Print C

Subroutine (Programmname: „Sub1")
A+B ⇒ C
Return

Beispiel 2: Springen in eine Subroutine bei Zuordnung von Werten zu den Parametervariablen der Subroutine

- In diesem Beispiel ordnet das Hauptprogramm Werte der Parametervariablen „E" in der mit „Sub1" benannten Subroutine sowie den Parametervariablen „F" und „G" in der mit „Sub2" benannten Subroutine zu.

Hauptprogramm
Input A
Input B
Sub1(A) ← Ordnet den Wert der Variablen „A" des Hauptprogramms der Parametervariablen (E) in der Subroutine „Sub1" zu und springt dann sofort zur Subroutine „Sub1".
Print C
Sub2(A,B) ← Ordner die Werte der Variablen „A" und „B" des Hauptprogramms den entsprechenden Parametervariablen (F und G) der Subroutine „Sub2" zu und springt dann sofort zur Subroutine „Sub2".
Print D

Subroutinenprogramm 1 (Programmname „Sub1")
E × 2 ⇒ C ← Erfordert die Eingabe des Variablennamens „E" in das Parametervariablenfeld.
Return

Subroutinenprogramm 2 (Programmname "Sub2")
F + G ⇒ D ← Erfordert die Eingabe der Variablennamen „F" und „G" in das Parametervariablenfeld.
Return

Tipp

- Die Subroutine muss sich nicht im aktuellen Ordner befinden. Um eine mit „Sub1" benannte Subroutine einzugeben, die sich zum Beispiel in dem mit „f1" benannten Ordner befindet, würden Sie „f1\Sub1( )" eingeben.

Beispiel 8:

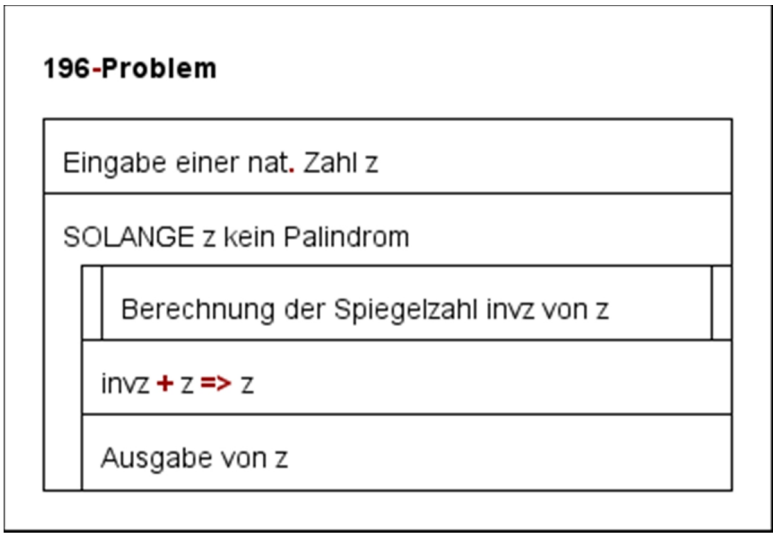
In diesem Beispiel wird ein Algorithmus formuliert, bei dem ein Unterprogramm mit Parameterübergabe verwendet wird.

Als Palindrom wird ein Wort genau dann bezeichnet, wenn es gleich bleibt, auch wenn man es von rechts nach links liest. Beispiele für Palindrome sind OTTO, ANNA oder RENTNER. Der Begriff des Palindroms wird auch auf Zahlen angewendet. Eine positive ganze Zahl wird genau dann Palindromzahl genannt, wenn sie mit ihrer Spiegelzahl (vergleiche Beispiel 7) übereinstimmt. So sind 6, 707, 7007 oder 69896 oder 698896 Palindromzahlen.

Nach folgender Vorschrift sollen Zahlenfolgen erzeugt werden:

1. Es wird eine natürliche Zahl z_0 gewählt.
2. Wenn z_k eine Palindromzahl ist, dann endet der Algorithmus.
3. Wenn z_k keine Palindromzahl ist, dann ergibt sich z_{k+1} aus der Summe von z_k und der Spiegelzahl von z_k.
4. Der Algorithmus wir bei 2. fortgesetzt.

Das Struktogramm für diesen Algorithmus:



Problematisch ist an diesem Algorithmus, dass bis heute nicht bekannt ist, ob er für jede am Anfang gewählte natürliche Zahl *z* nach endlich vielen Schritten stoppt.

Gegenwärtig ist 196 die kleinste natürliche Zahl, für die man nicht entscheiden kann, ob der angegebene Algorithmus nach endlich vielen Schritten endet.

(Vergleiche die Ausführungen von Prof. Jürgen Dankert auf http://www.rzbt.haw-hamburg.de/dankert/Palindrom/Zahlen-Palindrome/)

```
P196 | N
ClrText
Input z
SZUP(z)
While not(z=invz)
Print z
invz+z⇒z
Print invz
SZUP(z)
WhileEnd
Print z
```

Abb. 10 Hauptprogramm zum 196-Problem

```
SZUP | N | n
0⇒invz
Do
int(n/10)⇒rest
mod(n,10)⇒ziffer
10*invz+ziffer⇒invz
rest⇒n
LpWhile not(rest≤0)
```

Abb. 11 Unterprogramm zur Erzeugung der Spiegelzahl

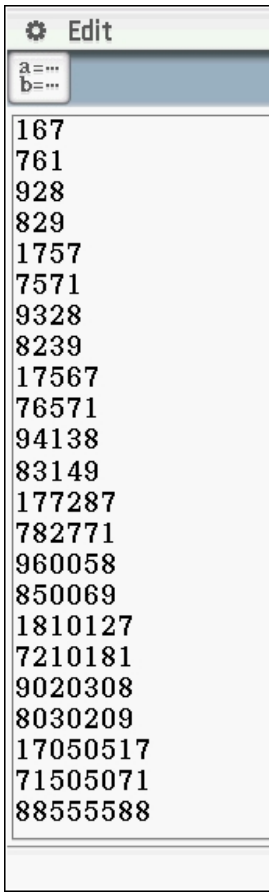


Abb. 12 Ergebnis für z=167

Hier wird die Realisierung des oben angegebenen 196-Algorithmus auf dem ClassPad II angegeben.

Wenn im Hauptprogramm **P196** für die Variable *z* der Wert 167 eingegeben wird, so erscheint das nebenstehende Ergebnis. Der Schleifenkörper des Hauptprogrammes wird 11-mal wiederholt, bis sich die Palindromzahl 88555588 ergibt.

Das Unterprogramm **SZUP(n)** erfordert beim Aufruf die Eingabe eines Parameterwertes für die Variable *n*. Die Variable *n* steht für die Zahl, von der die Spiegelzahl gebildet wird. Innerhalb des Unterprogramms erhält die Variable *invz* als Wert die Spiegelzahl von *n* zugeordnet.

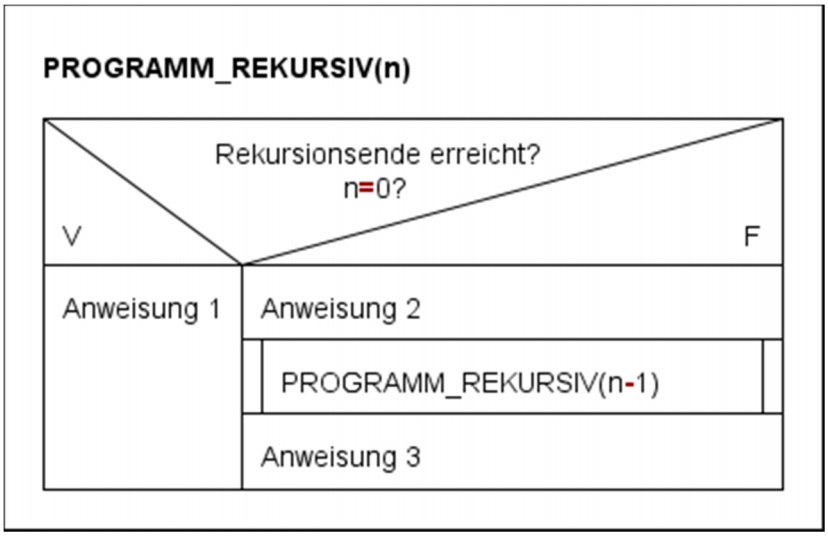
Die Variable *invz* wird bereits im Hauptprogramm verwendet, sie stellt somit eine globale Variable dar. Die Rückgabe von Variablenwerten, die im Unterprogramm berechnet werden, an das Hauptprogramm kann nur über globale Variable erfolgen.

4 Rekursion

Das Wesen rekursiver Programmierung besteht in der Erzeugung von Schleifen durch den Selbstaufruf von Funktionen oder Unterprogrammen.

Da in der Programmiersprache des ClassPad II das Definiens selbstdefinierter Funktionen nur aus genau einem arithmetischen Ausdruck bestehen darf, ist eine Realisierung von rekursiven Funktionen beim ClassPad II nicht möglich. Bleiben also nur rekursive Unterprogramme.

Das folgende Struktogramm zeigt den Aufbau eines rekursiven Unterprogramms.



Wird angenommen, dass der Aufruf dieses Unterprogramms vom Hauptprogramm durch den Befehl *PROGRAMM_REKURSIV(2)* erfolgt, dann wird folgendes geschehen:

- Da das Rekursionsende noch nicht erreicht ist, wird die Anweisung 2 ausgeführt;
- es erfolgt der Aufruf *PROGRAMM_REKURSIV(1);*
- die noch nicht ausgeführte Anweisung 3 wird in einem Stapelspeicher (Stack) abgelegt.
- Da das Rekursionsende noch nicht erreicht ist, wird die Anweisung 2 ausgeführt;
- es erfolgt der Aufruf *PROGRAMM_REKURSIV(0);*
- die noch nicht ausgeführte Anweisung 3 wird im Stack abgelegt. (Bis hierhin wird der Vorgang rekursiver Abstieg genannt, er endet mit dem Erreichen des Rekursionsendes.)
- Da jetzt das Rekursionsende (*n=0*) erreicht ist, wird Anweisung 1 ausgeführt.
- Anschließend wird der Stack nach dem LIFO-Prinzip (**l**ast **i**n **f**irst **o**ut) geleert. Die zuletzt abgelegte Anweisung 3 wird ausgeführt, danach die zuerst abgelegte Anweisung 3. (Dieser Vorgang wird rekursiver Aufstieg genannt.)

Somit wird die Anweisung 2 zweimal beim rekursiven Abstieg ausgeführt.

Die Anweisung 1 wird einmal beim Erreichen des Rekursionsendes ausgeführt und die Anweisung 3 wird zweimal beim rekursiven Aufstieg ausgeführt.

Unterprogramme dieser Art lassen sich mit dem ClassPad II verwirklichen.

Mit dem folgenden Unterprogramm wird genau dieses rekursive Verhalten demonstriert.

Beispiel 1 für Rekursion

Im Programm *Rekd* wurde für die Rekursionstiefe *a* der Wert 2 eingegeben .

Es ergab sich die rechts stehende Ausgabe.

Abb. 13 Hauptprogramm für Rekursionsdemonstration

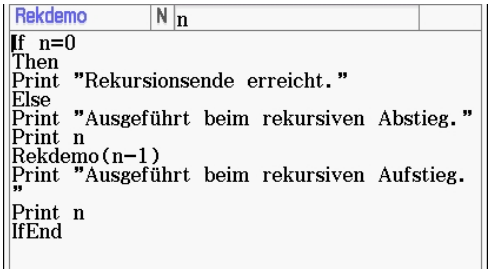


Abb. 14 Rekursives Unterprogramm

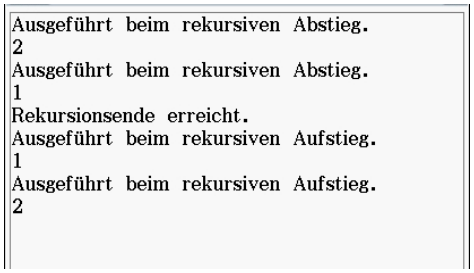


Abb. 15 Resultat für a=2

Bei der Eingabe der Rekurtionstiefe 39 für *a* ergibt sich erstmals die Fehlermeldung:

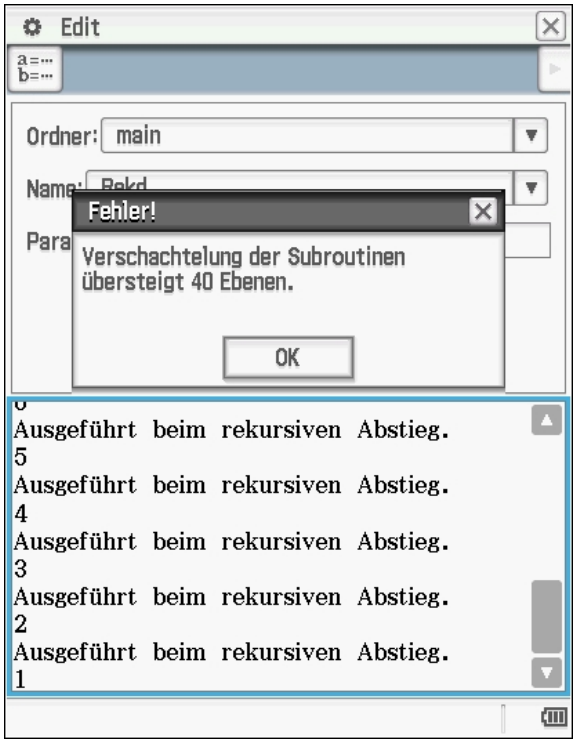


Abb. 16 Fehlermeldung bei Rekursion

Folglich darf die Rekursionstiefe beim ClassPad II 40 nicht übersteigen.

Beispiel 2 für Rekursion

Mit dem folgenden Unterprogramm *Kreism(d,n)* wird ein Muster von konzentrischen Kreisen von außen nach innen rekursiv erzeugt.

```
If n=0
Then
Return
Else
´circle 0,0,d*n
wait 2
´Kreism(d,n-1)
circle 0,0,d*n
IfEnd
```

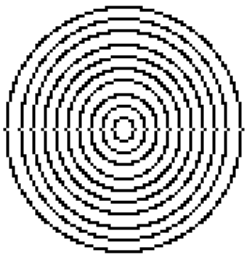


Abb. 17 Kreismuster

Der Aufruf des Unterprogramms erfolgt mit dem Hauptprogramm *Kmuster*

```
ClrGraph
Kreism(0.2,10)
```

Es entsteht von außen nach innen beim rekursiven Abstieg das nebenstehende Muster Abb. 17 Kreismuster.

Werden im Unterprogramm *Kreism(d,n)* die Befehlszeilen „`circle 0,0,d*n`“ und „`Kreism(d,n-1)`“ miteinander vertauscht, so entsteht dieselbe Figur, aber von innen nach außen. Nun werden die Kreise erst beim rekursiven Aufstieg gezeichnet.

Beispiel 3 für Rekursion

Mithilfe des Unterprogramms *Spivi(a,b,s,d,n)* kann eine Spirale von innen nach außen gezeichnet werden.

```
Spirale1 | N
ClrGraph
SetAxes Off
Spivi(-3,-1,1,0.3,8)
```

Abb. 18 Hauptprogramm für Spirale

```
Spivi | N | a, b, s, d, n
If n>0
Then
line a, b, a+s, b
Wait 1
line a+s, b, a+s, b+s+d
Wait 1
line a+s, b+s+d, a-d, b+s+d
Wait 1
line a-d, b+s+d, a-d, b-d
Wait 1
Spivi(a-d, b-d, s+2d, d, n-1)
IfEnd
```

Abb. 19 Rekursives UP für Spirale

Abb. 20 Spirale

Das Unterprogramm *Spiva(a,b,s,d,n)* entsteht aus dem Unterprogramm *Spivi(a,b,s,d,n)*, indem die Befehlszeilen zwischen `Then` und `IfEnd` in umgekehrter Reihenfolge geschrieben werden. Insbesondere steht der rekursive Aufruf nun unmittelbar hinter `Then`. Deshalb wird die Spirale nun von außen nach innen gezeichnet.

Beispiel 4 für Rekursion

Die rekursive Berechnung der Fakultät darf gewiss als das klassische Beispiel für Rekursion bezeichnet werden. Es soll auch hier nicht fehlen.

Als Definition für $n!$ (gelesen: n Fakultät) für nicht negative ganze Zahlen gelte

$$n! = \begin{cases} 1 & \text{für } n = 0 \\ (n-1)! \cdot n & \text{sonst} \end{cases}$$

Das folgende Unterprogramm verwendet genau diese Definition.

Werden für a Werte eingegeben, die größer als 38 sind, so erscheint eine Fehlermeldung, weil die Rekursionstiefe 40 übersteigt.

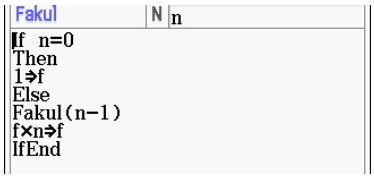


Abb. 21 UP für n!

Abb. 22 HP für n!

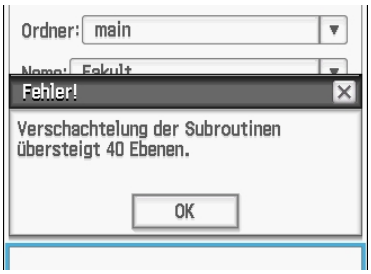


Abb. 23 Fehleranzeige

Beispiel 5 für Rekursion

Als Quersumme einer natürlichen Zahl wird die Summe ihrer Ziffern bezeichnet.

Mit

$$q(n) = \begin{cases} n & \text{, wenn } n < 10 \\ \text{Einerziffer} + q(n \text{ ohne Einerziffer}), & \text{sonst} \end{cases}$$

oder mit ClassPad-Befehlen

$$q(n) = \begin{cases} n & \text{, wenn } n < 10 \\ \text{mod}(n,10) + q(\text{int}(n/10)) & \text{sonst} \end{cases}$$

wird die Quersumme einer natürlichen Zahl n rekursiv definiert.

Das folgende Unterprogramm *qs(n)* berechnet die Quersumme einer Zahl, die im Hauptprogramm *quer* eingegeben wird

qs(n):

```
If n≥1
Then
q+mod(n,10)⇒q
int(n/10)⇒a
qs(a)
IfEnd
```

quer:

```
ClrText
0⇒q
Input a
qs(a)
Print q
```

Beispiel 6 für Rekursion

Ähnlich wie die Berechnung der Fakultät einer natürlichen Zahl gehört auch die Lösung der Aufgabe „Turm von Hanoi“ zu den Standardaufgaben der Rekursion.

Das Spiel „Turm von Hanoi“ geht zurück auf den französischen Mathematiker ÉDOUARD LUCAS, der es 1883 erfunden haben soll.

Die Aufgabe besteht darin, einen Turm aus *n* auf dem Platz *A* übereinander liegenden Scheiben, deren Durchmesser von unten nach oben abnimmt, mit möglichst wenigen Zügen so zum Platz *B* umzustapeln, dass jeweils nur eine Scheibe bewegt wird und nie eine kleinere Scheibe auf einer größeren liegt. Dabei darf ein Hilfsplatz *H* benutzt werden. Ein Zug besteht darin, dass eine obere Scheibe von einem Platz zu einem anderen gelegt wird.

Für eine Scheibe besteht die Lösung in einem Zug; die Scheibe wird von A nach B gelegt.

Für zwei Scheiben sind genau drei Züge erforderlich:

- obere Scheibe von *A* nach *H*,
- größte Scheibe von *A* nach *B*,
- Scheibe von *H* nach *B*.

Wenn *n* (*n*>1) Scheiben auf dem Platz *A* liegen, dann führt der folgende rekursive Ansatz zur Lösung.

1. Man baue den Turm, der aus den oberen $n-1$ Scheiben besteht, mit möglichst wenigen Zügen von *A* nach *H* um.
2. Man lege die größte Scheibe von A nach B.
3. Man baue den Turm aus $n-1$ Scheiben mit möglichst wenigen Zügen von *H* nach *B* um.

Es lässt sich zeigen, dass $2^n - 1$ die geringste Anzahl von Zügen ist, die für den Umbau eines Turmes aus n Scheiben von *A* nach *B* erforderlich sind.

Mit dem rekursiven Unterprogramm *umbau(n,A,B,H)* wird diese Lösungsstrategie umgesetzt.

Die Parameter haben die folgenden Bedeutungen:

n: Anzahl der Scheiben

A: Platz, auf dem am Anfang der Turm aus *n* Scheiben steht. Im Hauptprogramm erhält *a* den Wert 1.

B: Platz, auf dem der Turm aus *n* Scheiben am Ende stehen soll. Im Hauptprogramm erhält b den Wert 2.

H: Hilfsplatz, der beim Umbau benutzt werden darf. Im Hauptprogramm erhält *h* den Wert 3.

Ausgegeben werden die einzelnen Züge als zweistellige Zahlen in ihrer richtigen Reihenfolge.

So bedeutet zum Beispiel 13, dass bei diesem Zug die oberste Scheibe vom Platz 1 (*A*) zum Hilfsplatz 3 (*H*) gelegt wird.

umbau(n,A,B,H):

```
IF n=1
Then
Print 10a+b
Else
umbau(n-1,a,h,b)
Print 10a+b
umbau(n-1,h,b,a)
IfEnd
```

Der Aufruf des Unterprogramms *umbau(n,A,B,H)* erfolgt durch das Hauptprogramm

Turm:

```
ClrText
Input n,"Scheibenanzahl","Scheibenanzahl"
umbau(n,1,2,3).
```

Wird im Hauptprogramm für $n = 4$ eingegeben, so ergibt sich:

```
umbau | N | n, A, B, H
If n=1
Then
Print 10×A+B
Else
umbau(n-1, A, H, B)
Print 10×A+B
umbau(n-1, H, B, A)
IfEnd
```

Abb. 24 HP für Türme

```
Turm | N |
ClrText
Input n, "Scheibenanzahl", "Scheibe
nanzahl"
umbau(n, 1, 2, 3)
```

Abb. 25 Rek. UP für Türme

```
13
12
32
13
21
23
13
12
32
31
21
32
13
12
32
```

Abb. 26 Ergebnis für n=4

Wird in das Hauptprogramm und das Unterprogramm eine Zählvariable `zuege` in der angegebenen Weise eingefügt, so wird auch noch die Anzahl der erforderlichen Züge ausgegeben.

```
IF n=1
Then
Print 10A+B
zuege+1⇒zuege
Else
umbau(n-1,A,H,B)
Print 10A+B
zuege+1⇒zuege
umbau(n-1,H,B,A)
IfEnd
```

```
ClrText
Input n,"Scheibenanzahl",
"Scheibenanzahl"
0⇒zuege
umbau(n,1,2,3)
Print zuege
```

Beispiel 7 für Rekursion

Das Bisektionsverfahren beruht auf dem Satz, dass jede stetige Funktion, deren Funktionswerte an den Rändern eines abgeschlossenen Intervalls unterschiedliche Vorzeichen haben, in diesem Intervall mindestens eine Nullstelle aufweist. Das ist eine Folgerung aus dem Zwischenwertsatz für stetige Funktionen.

Hat man für eine stetige Funktion f Zahlen a und b mit den Eigenschaften $a < b$ und $f(a) \cdot f(b) < 0$ gefunden, so kann man die Lage einer Nullstelle von f präzisieren, indem man das Intervall $a \le x \le b$ halbiert und feststellt, ob eine Nullstelle von f im

Intervall $a \le x \le \frac{a+b}{2}$ oder im Intervall $\frac{a+b}{2} \le x \le b$ liegt. Wenn $f(a)$ und $f\left(\frac{a+b}{2}\right)$ unterschiedliche Vorzeichen haben, also $f(a) \cdot f\left(\frac{a+b}{2}\right) < 0$, dann liegt eine Nullstelle von f im Intervall $a \le x \le \frac{a+b}{2}$, sonst liegt eine Nullstelle im Intervall $\frac{a+b}{2} \le x \le b$. Das gefundene Intervall wird nun wieder halbiert, und es wird festgestellt, in welcher Hälfte dieses Intervalls eine Nullstelle von f liegt. Dieses Vorgehen wird (rekursiv) fortgesetzt, bis die Intervalllänge hinreichend klein ist (das Rekursionsende erreicht ist), also eine Nullstelle mit hinreichender Genauigkeit ermittelt wurde.

Am Beispiel der Funktion $fun(x) = x^x - 3$ soll dieses rekursive Vorgehen verwirklicht werden.

Da $fun(1) = -2$ und $fun(2) = 1$ unterschiedliche Vorzeichen haben, muss sich im Intervall $1 \le x \le 2$ eine Nullstelle von fun befinden.

Wird bei der Ausführung des Hauptprogramms $a = 1$, $b = 2$ und $g = 10^{-10}$ eingegeben, so wird als Näherungswert der Nullstelle $x_0 \approx 1{,}82545507$ berechnet.

Hauptprogramm **NST**:

```
ClrText
Define fun(x)=x^x-3
Input a,"","linkes
Intervallende"
Input b,"","rechtes
Intervallende"
Input g,"","Genauigkeit"
bisekt(a,b,g)
Print x0
```

Unterprogramm **bisekt(a,b,g)**:

```
If b-a<g
Then
(a+b)/2⇒x0
ElseIf fun(a)×fun((a+b)/2)<0
Then
(a+b)/2⇒b
bisekt(a,b,g)
Else
(a+b)/2⇒a
bisekt(a,b,g)
IfEnd
```

Beispiel 8 für Rekursion

Fraktale sind Beispiele für Figuren, die rekursiv erzeugt werden können.

Selbstähnlichkeit ist das Wesensmerkmal von Fraktalen.

Ein bekanntes und gründlich studiertes Fraktal ist das Sierpinski-Dreieck, das nach dem polnischen Mathematiker WACLAW SIERPINSKI (1882 - 1969) benannt wurde.

Die Selbstähnlichkeit des Sierpinski-Dreiecks ergibt sich sofort aus seiner Entstehung:

1. Ein Dreieck als Ausgangsfigur wird durch seine drei Mittellinien in vier kongruente Dreiecke aufgeteilt.

2. Das mittlere Dreieck wird herausgenommen.

3. Dieser Vorgang wird ständig wiederholt (Rekursion).

Hier wird als Ausgangsdreieck ein gleichschenkliges Dreieck gewählt, dessen Eckpunkte (x/y), $(x+s/y)$ und $\left(x+\frac{s}{2}/y+s\right)$ sind. Die Basis und die Höhe auf der Basis dieses Dreiecks haben somit die Länge s.

Beim Zeichnen des Musters hilft folgende Übersicht.

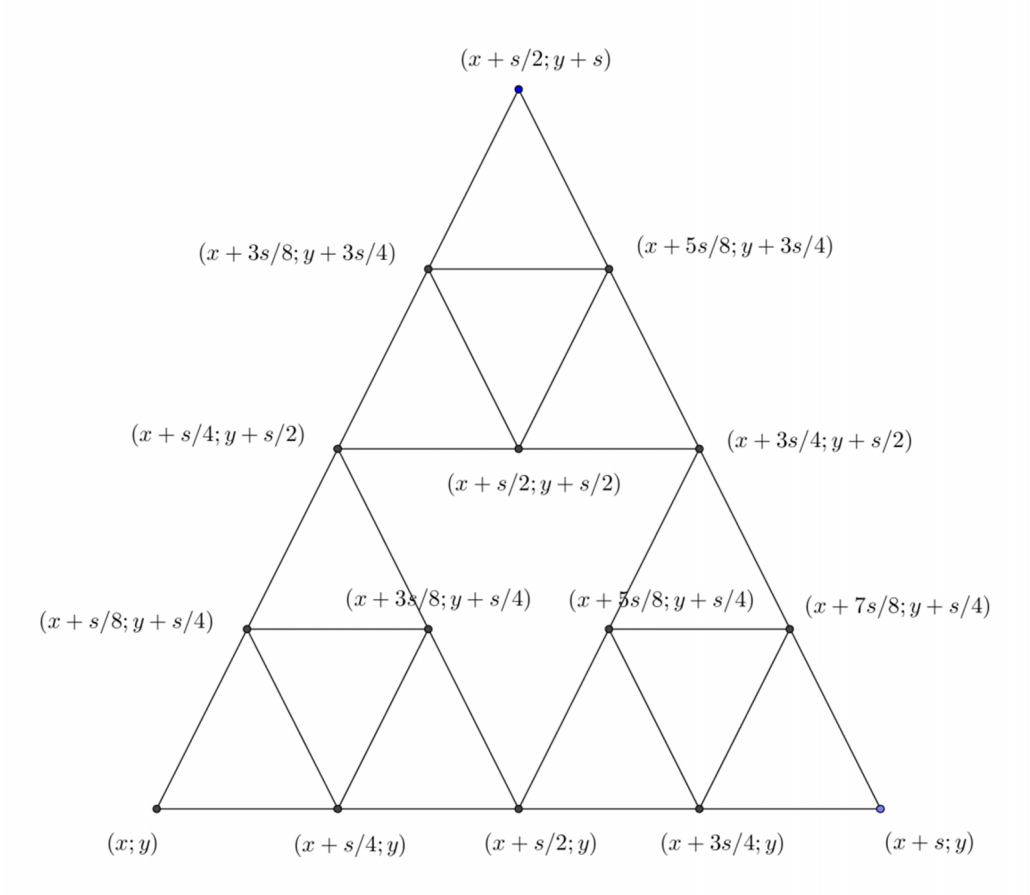


Abb. 27 Koordinaten beim Sierpinski-Dreieck

SDreieck:

```
ClrGraph
-2⇒x
-2.5⇒y
6⇒s
Input n,"","Rekursionstiefe"
line x+s/4,y+s/2,x+s/2,y
line x+s/2,y,x+3/4*s,y+s/2
line x+3/4*s,y+s/2,x+s/4,y+s/2
Muster(x,y,s,n)
```

Muster(x,y,s,n):

```
If n=1
Then
line x+s/8,y+s/4,x+s/4,y
line x+s/4,y,x+3×s/8,y+s/4
line x+s/8,y+s/4,x+3×s/8,y+s/4

line x+5×s/8,y+s/4,x+3×s/4,y
line x+3×s/4,y,x+7×s/8,y+s/4
line x+7×s/8,y+s/4,x+5×s/8,y+s/4

line x+3×s/8,y+3×s/4,x+s/2,y+s/2
line x+s/2,y+s/2,x+5×s/8,y+3×s/4
line
x+5×s/8,y+3×s/4,x+3×s/8,y+3×s/4
Else
Muster(x,y,s/2,n-1)
Muster(x+s/2,y,s/2,n-1)
Muster(x+s/4,y+s/2,s/2,n-1)
IfEnd
```

Wird im Hauptprogramm für $n = 6$ eingegeben, so ergibt sich folgendes Bild.

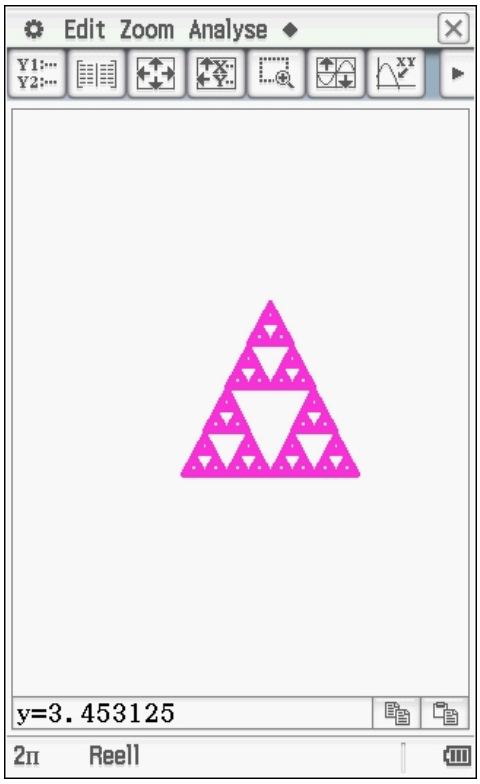


Abb. 28 Sierpinski Dreieck für n=6

5. Programme für den Mathematikunterricht

5.1 Würfeln mit dem ClassPad II

Erzeugung einer Verteilung

Wenn man das Würfeln simulieren möchte, benutzt man dazu in der Regel die Tabellenkalkulation. Bzgl. des ClassPads in der Version 330[1] gibt es hier aber Probleme, da der Speicher der Tabellenkalkulation begrenzt ist. Dies gilt auch, wenn man mit dem ClassPad Manager arbeitet. Der Grund für die Begrenztheit des Speichers ist darin begründet, dass für die Tabellenkalkulation ebenfalls das CAS zur Verfügung steht; so lassen sich zum Beispiel auch Tabellen für Ableitungen erstellen oder näherungsweise berechnete Integralwerte in Abhängigkeit der oberen Grenze.

Die Anzahl der geworfenen Augenzahlen werden in den Variablen *qa*, *qb*, ... und *qf* gespeichert. Alternativ hätte man die Zahlen auch in einer Liste ablegen können; dies ist aber nicht unbedingt erforderlich, da man in der Tabellenkalkulation auch direkt auf die Variablen zugreifen kann.

Der Befehl *Print n* in der vorletzten Zeile ist für den Ablauf natürlich nicht erforderlich; beruhigt aber den Anwender, da die Erzeugung der Wurfergebnisse doch eine gewisse Zeit in Anspruch nimmt.

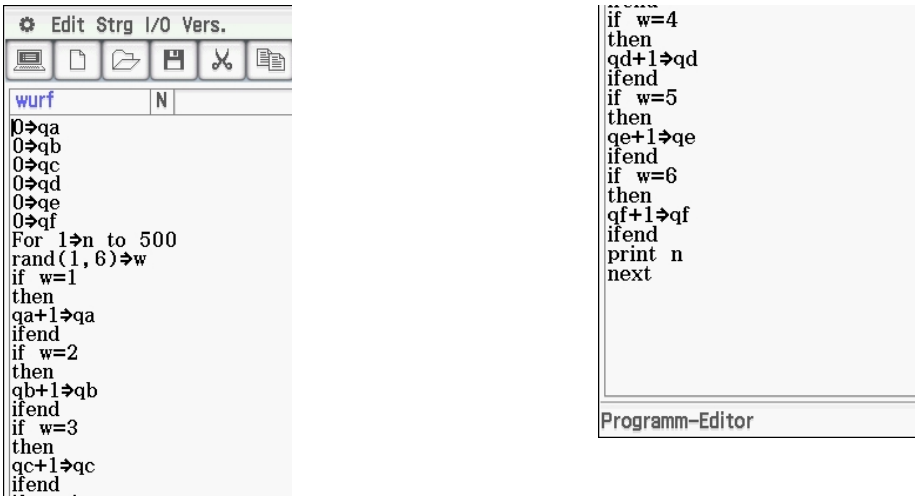


Abb. 29 Simulation von 500 Würfeln

[1] Für die neue Version ClassPad II wurde der Speicher für die Tabellenkalkulation erheblich vergrößert, so dass dieses Problem nicht mehr entsteht.

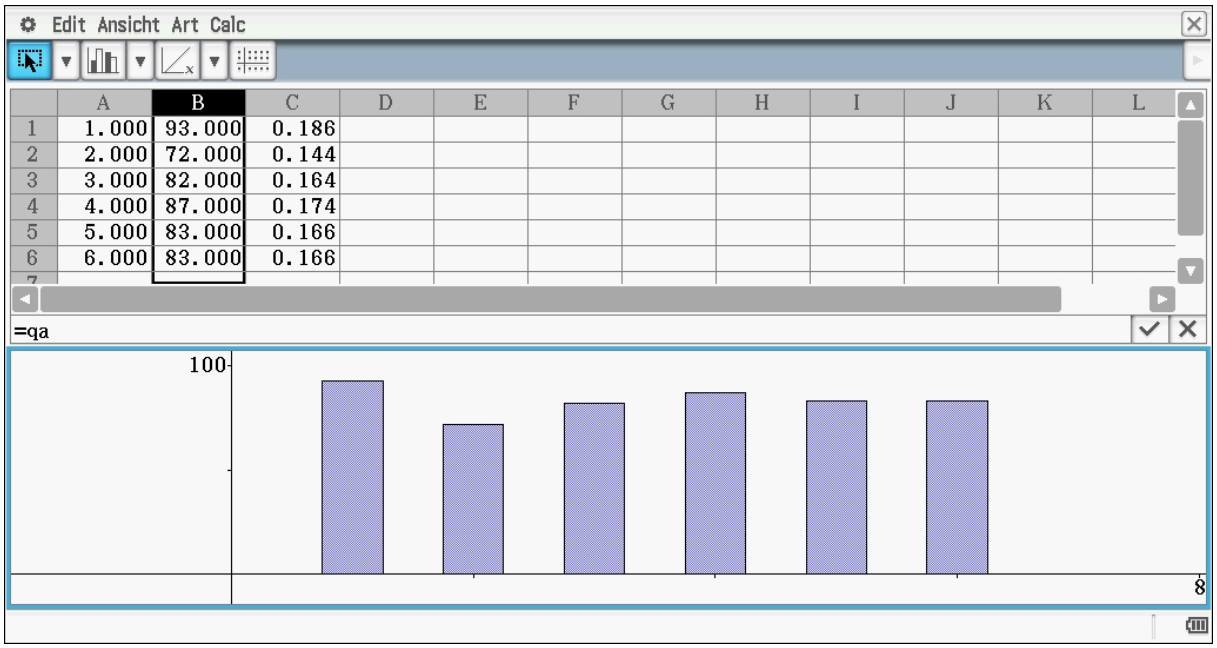


Abb. 30 Darstellung der Würfelergebnisse mit der Tabellenkalkulation

Wie lange muss man im Durchschnitt warten, bis man eine „6“ würfelt?

Um diese Frage zu beantworten, lässt sich natürlich der Erwartungswert berechnen. Im Vergleich dazu ist es natürlich interessant, eine Simulation durchzuführen.

Das Programm ist in Abbildung 31 dargestellt. Man beachte, dass hier eine „While“-Schleife benutzt wird, da die Anzahl der Würfe, die erforderlich sind, natürlich nicht vorher festgelegt werden kann. Um die Werte anschließend weiter zu verarbeiten, werden sie in Listen gespeichert.

Als Bedingung für die *While*-Schleife ist natürlich auch *not(w=6)* möglich. Entsprechendes gilt für die *If*-Abfrage.

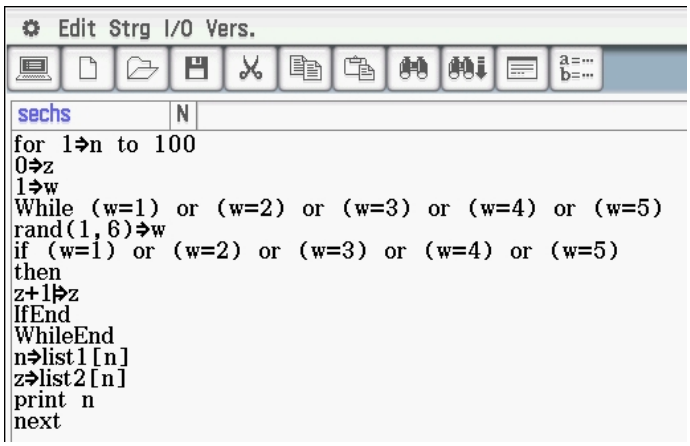
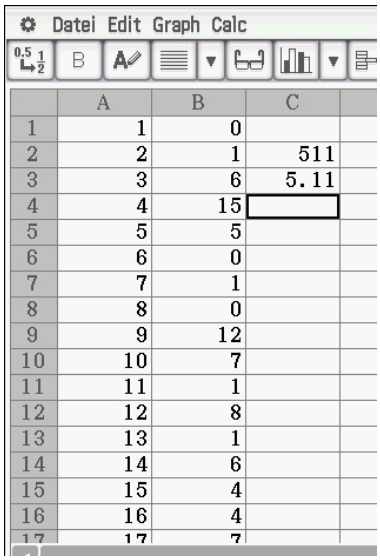


Abb. 31 Programm zur Ermittlung der Würfe, die für eine "6" benötigt werden



	A	B	C
1	1	0	
2	2	1	511
3	3	6	5.11
4	4	15	
5	5	5	
6	6	0	
7	7	1	
8	8	0	
9	9	12	
10	10	7	
11	11	1	
12	12	8	
13	13	1	
14	14	6	
15	15	4	
16	16	4	
17	17	7	

Abb. 32 Die Werte der ersten 17 Versuche

Mit Hilfe der Tabellenkalkulation lässt sich der Durchschnittswert (s. Zelle C3, Abb. 32) leicht ermitteln[2]. Der Wert lässt sich natürlich auch direkt bestimmen. In der Tabelle unten sind die Wahrscheinlichkeiten für die Wahrscheinlichkeiten dafür angegeben, dass beim ersten, zweiten usw. Mal eine „6“ gewürfelt wird.

Anzahl	1	2	3	4
Wahrscheinlichkeit	$\frac{1}{6}$	$\frac{5}{6}\cdot\frac{1}{6}$	$\frac{5}{6}\cdot\frac{5}{6}\cdot\frac{1}{6}$	$\left(\frac{5}{6}\right)^3\cdot\frac{1}{6}$

Verallgemeinert man die Werte der obigen Tabelle, so ergibt sich für den Erwartungswert:

$$E = \frac{1}{6}\cdot\sum_{n=1}^{\infty} n\cdot\left(\frac{5}{6}\right)^{n-1}.$$

Bei dem Ausdruck handelt es sich um eine abgewandelte geometrische Reihe. Der Wert lässt sich berechnen, wenn man den üblichen Trick zur Bestimmung des Grenzwertes einer geometrischen Reihe 2-mal anwendet. Dank des CAS lässt sich der Grenzwert aber direkt mit dem ClassPad II bestimmen, wie die Abbildung 33 zeigt. Man erhält den Erwartungswert 6, den man natürlich auch ohne Berechnungen und Simulationen erwartet hätte.

Abb. 33 Bestimmung des Grenzwertes der Reihe

[2] Dies erreicht man mit dem Befehl: *=mean(B1:B17).*

Das „Raab“-Problem

In der Fernsehsendung „Schlag den Raab“ gibt es das folgende Würfelspiel. Die Augenzahlen werden solange aufaddiert, bis eine 6 gewürfelt wird. Nach der obigen Untersuchung wissen wir, dass der Erwartungswert für die Anzahl der Würfe, bis die erste 6 erscheint den Wert 6 hat. Da man im Schnitt eine 3 würfelt, müsste die Summe 18 betragen; bzw. 15, wenn man rechtzeitig aufhören will. Durch eine kleine Veränderung des obigen Programms, lässt sich dies durch eine Simulation überprüfen.

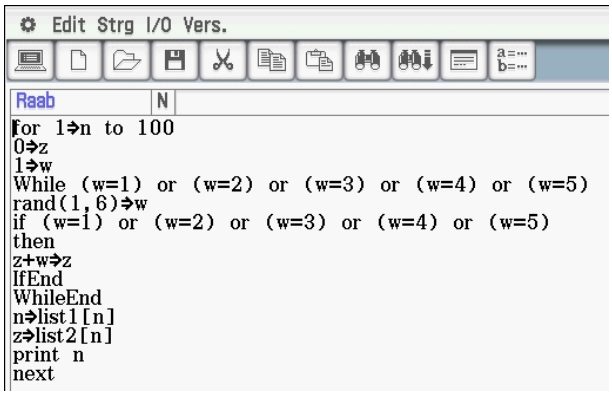
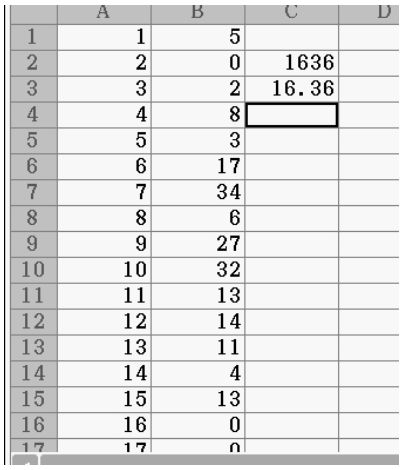


Abb. 34 Das Spiel aus "Schlag den Raab"

Die Werte lassen sich mit Hilfe der Tabellenkalkulation weiter bearbeiten.



	A	B	C	D
1	1	5		
2	2	0	1636	
3	3	2	16.36	
4	4	8		
5	5	3		
6	6	17		
7	7	34		
8	8	6		
9	9	27		
10	10	32		
11	11	13		
12	12	14		
13	13	11		
14	14	4		
15	15	13		
16	16	0		
17	17	0		

Abb. 35 Die Ergebnisse der Raab-Simulation

Der simulierte Erwartungswert steht in der Zelle C3 (Abb. 35)[3] und passt ganz gut zum obigen theoretischen Wert.

[3] Der Erwartungswert wurde durch Summation der Werte der Spalte B bestimmt (C2). Die Summe wurde dann durch 100 dividiert, da 100 Simulationen vorgenommen worden sind (C3).

Das Problem der vollständigen Serie

Vor allem bezüglich von Sammelbildern stellt sich die Frage, wie viele Packungen man im Schnitt kaufen muss, um eine vollständige Serie zu erhalten. Wir führen die Simulation bezogen auf ein Würfelmodell durch, das heißt, wie oft muss man im Durchschnitt würfeln, damit alle Zahlen mindestens einmal vorgekommen sind. Mit dem folgenden Programm wird eine Serie von 100 Versuchen realisiert.

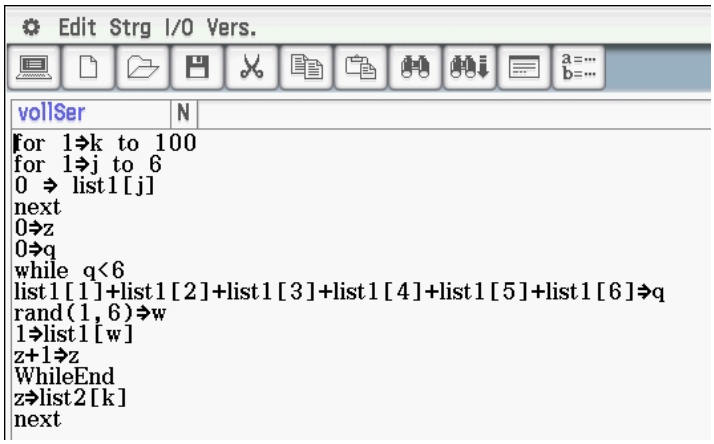
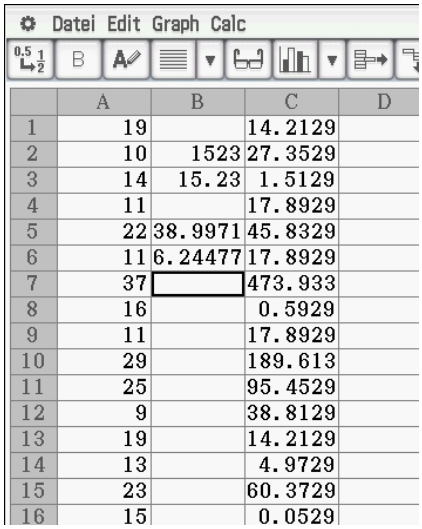


Abb. 36 Programm zur Simulation vollständiger Serien

Die erhaltenen Anzahlen, bis jede Zahl mindestens einmal vorgekommen ist, werden in einer Liste gespeichert und können so zum Beispiel mit der Tabellenkalkulation weiter bearbeitet werden. Die Werte bekommt man mit Hilfe der *Import*-Funktion[4] in die erste Spalte der Tabellenkalkulation. In der dritten Spalte stehen die Quadrate der jeweiligen Abweichungen. So hat man z.B. die Möglichkeit neben dem Durchschnittswert auch noch die Standardabweichung berechnen zu lassen. (s. folgende Abbildung)

Der Durchschnittswert steht in B3, die Standardabweichung in B6 (s. Abb. 37)



	A	B	C	D
1	19		14.2129	
2	10	1523	27.3529	
3	14	15.23	1.5129	
4	11		17.8929	
5	22	38.9971	45.8329	
6	11	6.24477	17.8929	
7	37		473.933	
8	16		0.5929	
9	11		17.8929	
10	29		189.613	
11	25		95.4529	
12	9		38.8129	
13	19		14.2129	
14	13		4.9729	
15	23		60.3729	
16	15		0.0529	

B2: = *sum(A1:A100)*

B3: = *B2/100*

C1: = *(A1-B3)^2*

C2: = *(A2-B3)^2*

usw.

B5: = *sum(C1:C100)*

B6: = *B5^0.5*

Abb. 37 Mögliche Werte der Simulation für eine vollständige Serie

[4] Die *Import*-Funktion findet man im Menü Datei

Auch hier lässt sich der Erwartungswert natürlich direkt bestimmen. Dazu betrachten wir zunächst nochmals die Frage, wie oft man in der Regel würfeln muss, bis eine „6“ erscheint. Zur Lösung erstellen wir ein Ablaufdiagramm.

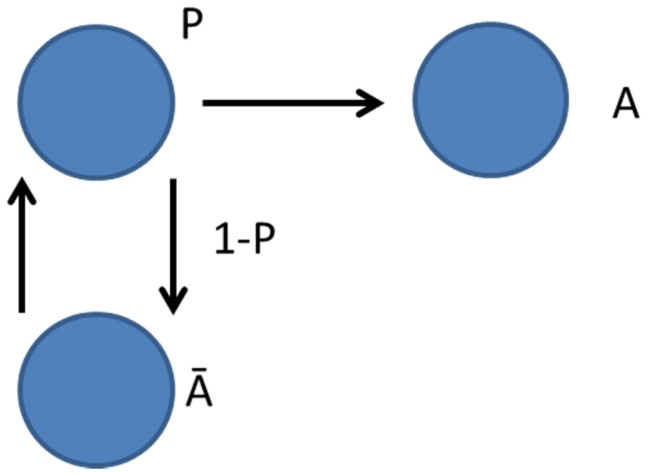


Abb. 38 Diagramm "Anzahl der Würfe, bis eine "6" erscheint

Das gesuchte Ereignis A tritt mit der Wahrscheinlichkeit p ein. Falls das Ereignis nicht eingetreten ist, befindet man sich wieder in der Ausgangssituation. Daraus folgt:

$$E(X) = 1 \cdot p + (1-p) \cdot (1 + E(X)) \;\Rightarrow\; E(X) = \frac{1}{p} .$$

Dieses lässt sich auf unser Problem übertragen.

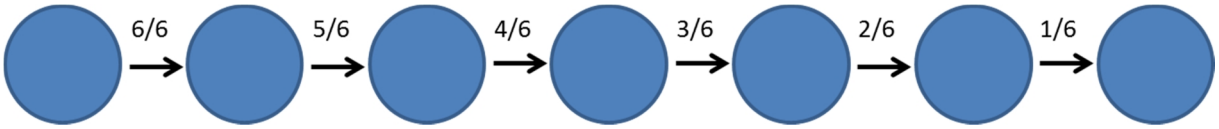


Abb. 39 Kette für die vollständige Serie

Für den ersten Wurf ist es vollkommen egal, welche Zahl gewürfelt wird. Beim zweiten Wurf darf nur die erste Zahl nicht wieder gewürfelt werden. Ist dies der Fall bleibt man in dem Zustand. Das heißt, man kann die Gleichung (*) auf jede Position anwenden. Wegen der Linearität der Erwartungswerte sind diese zu addieren.

Daraus folgt:

$$\begin{aligned} E(X) &= 1 + E(X_2) + E(X_3) + E(X_4) + E(X_5) + E(X_6) \\ &= 1 + \frac{6}{5} + \frac{6}{4} + \frac{6}{3} + \frac{6}{2} + \frac{6}{1} = 14{,}7 \end{aligned}$$

Das Ergebnis zeigt, dass unser Wert größenordnungsmäßig stimmt.

5.2 Wachstumsprozesse

Im Folgenden sollen die verschiedenen Wachstumsarten mit Hilfe von „Würfelexperimenten“ simuliert werden.

Lineares Wachstum

Lineares Wachstum ist dadurch ausgezeichnet, dass in gleichen Zeiten immer dasselbe hinzukommt. Dies lässt sich mit Hilfe von Würfeln zum Beispiel dadurch simulieren, dass mit 50 Würfeln jeweils die Anzahl der „6“ zählt und diese Zahlen aufaddiert. Die Abbildung 40 zeigt das entsprechende Programm.

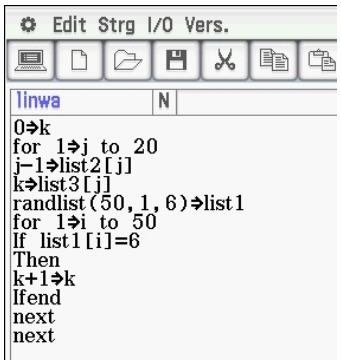


Abb. 40 Programm zur Simulation des linearen Wachstums

Die gewonnenen Daten werden in der Liste 3 und, um eine Zuordnung zu erhalten, werden die Ordnungszahlen des Wurfes in der Liste 2 gespeichert. Man hat jetzt die Möglichkeit, die Daten im Statistik-Modul und mit der Tabellenkalkulation weiter zu bearbeiten. In der Statistik lässt sich dann auch eine lineare Regression durchführen. Die Abbildung 41 zeigt das Ergebnis.

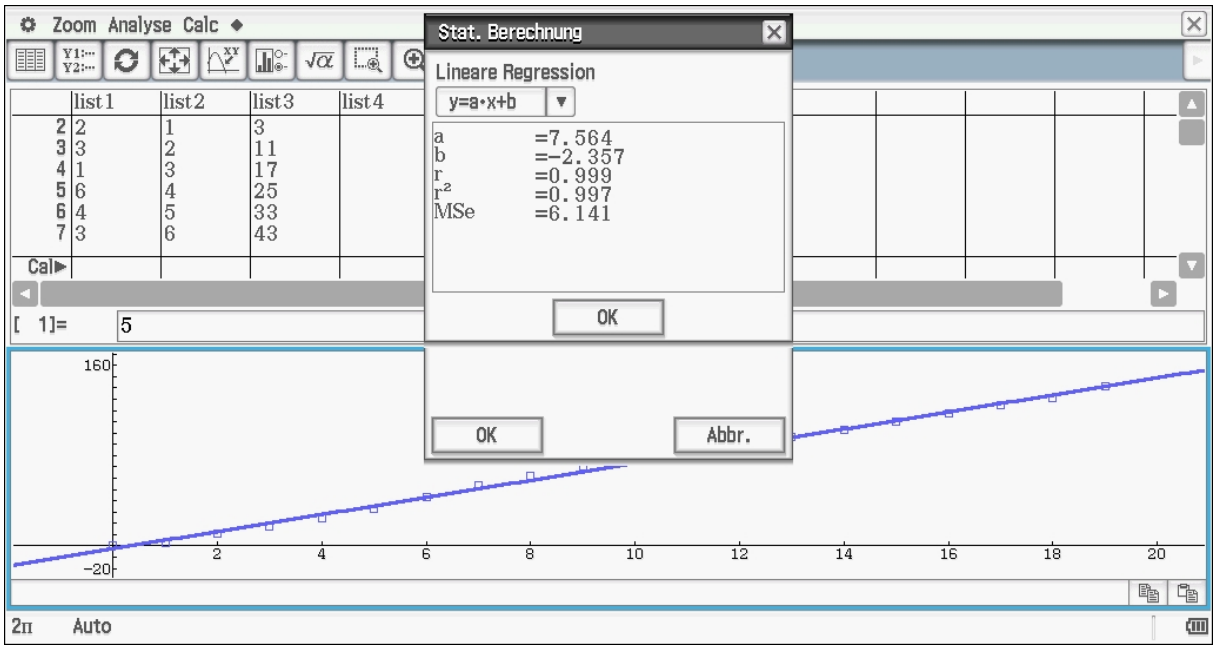


Abb. 41 Bearbeitung der Daten im Statistik-Modul

Lineares Wachstum wird in der Schule eher in Klasse 7 oder 8 diskutiert. Von daher ist die Durchführung einer Regression sicher nicht altersgemäß, und man sollte die Tabellenkalkulation heranziehen. Die entsprechenden Werte findet man in den Spalten A und B (s. Abb. 42). Der Zuwachs ist in Spalte C berechnet. In D3 findet man den durchschnittlichen Zuwachs und in E3 den theoretischen Wert $\frac{50}{6}$.

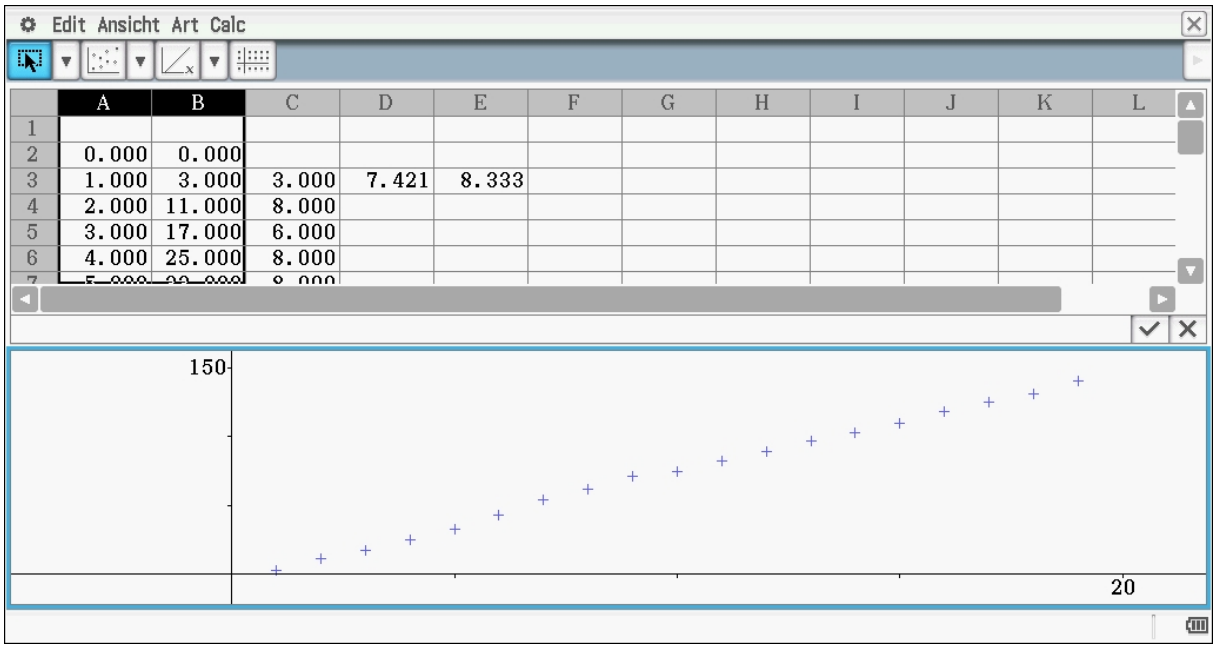


Abb. 42 Bearbeitung der Daten des linearen Wachstums mit der Tabellenkalkulation

Exponentielles Wachstum

Zur Einführung des exponentiellen Wachstums kann man zum Beispiel von 50 Würfeln ausgehen. Mit diesen wird gewürfelt, und die Anzahl der gewürfelten „6“ festgestellt. Waren zum Beispiel 8 x „6“ dabei, so wird die Anzahl der Würfel um diese 8 erhöht; das heißt, man würfelt das nächste Mal mit 58 Würfeln usw. Dieser Vorgang lässt sich natürlich auch mit Hilfe eines entsprechenden Programms simulieren.

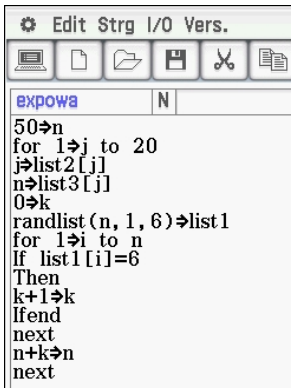
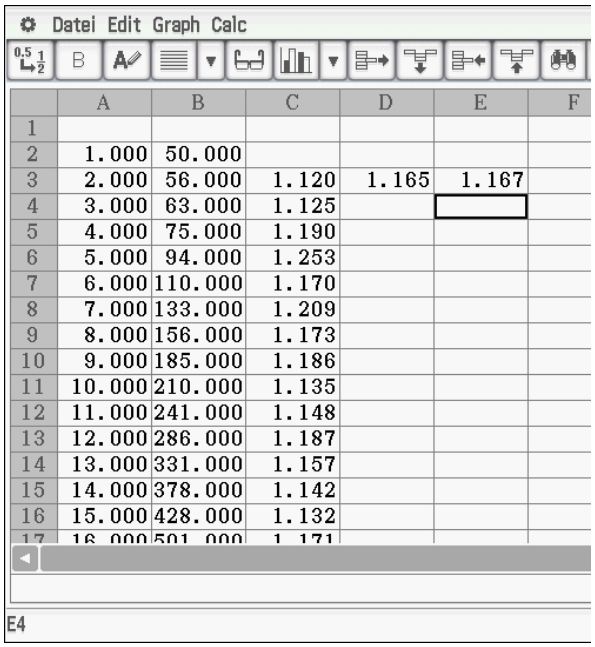


Abb. 43 Programm für die Simulation des exponentiellen Wachstums

Die Abbildung 43 zeigt das Programm. Der Vorgang des Würfeln findet 20 mal statt (2. Zeile) Zur Weiterverarbeitung werden die Daten in Listen gespeichert (Zeile 3 u. 4) Der Vorgang des Würfelns wird durch den Befehl *randlist(n,1,6)* realisiert. Dabei steht das *n* für die Anzahl der Würfe, die „1“ und die „6“ geben an, dass Zufallszahlen zwischen 1

und 6 erzeugt werden. Mit Hilfe der *If*-Schleife wird die Anzahl der „6“ festgestellt und die Zahl wird in der Variablen *k* gespeichert. Der Befehl *Next* markiert das Ende der *For*-Schleife. Die so gewonnenen Daten stehen jetzt in den Listen *list2* und *list3*. Sie stehen im Statistik-Modul direkt zur Weiterverarbeitung bereit. Hier ist es aber von Vorteil, für die Weiterverarbeitung die Tabellenkalkulation zu benutzen. Über die Importfunktion im Untermenü *Datei* lassen sich die Daten direkt in die Tabellenkalkulation übertragen.



	A	B	C	D	E	F
1						
2	1.000	50.000				
3	2.000	56.000	1.120	1.165	1.167	
4	3.000	63.000	1.125			
5	4.000	75.000	1.190			
6	5.000	94.000	1.253			
7	6.000	110.000	1.170			
8	7.000	133.000	1.209			
9	8.000	156.000	1.173			
10	9.000	185.000	1.186			
11	10.000	210.000	1.135			
12	11.000	241.000	1.148			
13	12.000	286.000	1.187			
14	13.000	331.000	1.157			
15	14.000	378.000	1.142			
16	15.000	428.000	1.132			
17	16.000	501.000	1.171			

Abb. 44 Ergebnisse des Würfelns

In der Spalte C sind die Quotienten zweier aufeinander folgender Anzahlen berechnet[5]. Man erkennt, dass diese näherungsweise konstant sind und sich in etwa der Wert $\frac{7}{6}$ ergibt, was ja auch nicht anders zu erwarten war. Der durchschnittliche Quotient steht in D3 (= *mean(C3:C52)*) und die Näherung von $\frac{7}{6}$ in E3.

Wenn man diese Simulation mit einem Handheld durchführt, werden ca. 25 Minuten benötigt. Mit dem ClassPad Manager dauert dies 2 bis 3 Minuten.

Durch eine kleine Abänderung im Programm lässt sich natürlich auch statt des exponentiellen Wachstums der exponentielle Zerfall simulieren. Dafür ist die Ausgangszahl zu erhöhen und in der vorletzten Zeile das „+“ durch ein „–“ zu ersetzen.

Beschränktes Wachstum

Für das beschränkte Wachstum gibt es eine Grenze, wie der Name schon sagt. Der jeweilige Zuwachs ist dann proportional zur Differenz zwischen dieser Grenze und der gegenwärtigen Anzahl. Für die Simulation setzen wir als willkürliche Grenze 500 und beginnen mit 50. Wir führen die Simulation wieder 20 mal durch. Die Abbildung 45 zeigt das entsprechend veränderte Programm. Man erkennt, dass nur *n* durch 500-*n* ersetzt worden ist.

[5] C3: = B3/B2

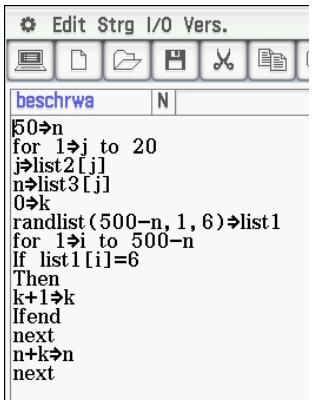


Abb. 45 Simulation beschränktes Wachstum

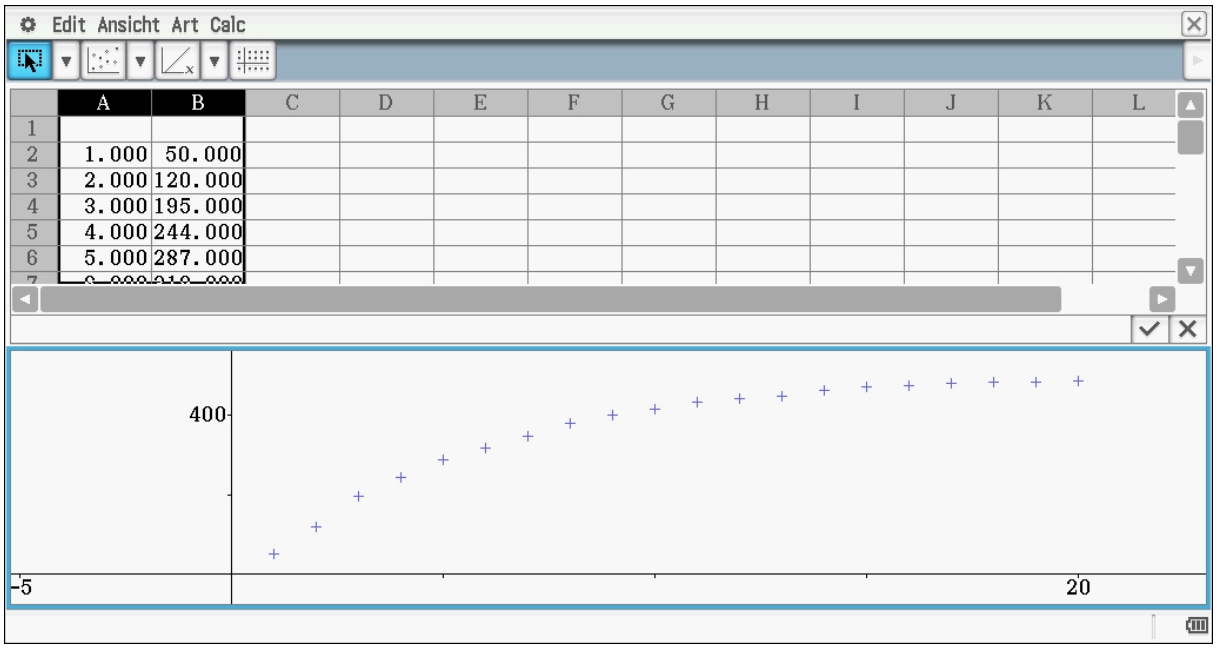


Abb. 46 Veranschaulichung der Daten des beschränkten Wachstums

Die Abbildung 46 zeigt den typischen Graphen für das beschränkte Wachstum.

Das logistische Wachstum

Hier sind die Verhältnisse komplizierter, was daran liegt, dass der Zuwachs sowohl zum Bestand, als auch zur Differenz zwischen Bestand und Beschränkung proportional ist. Es ist leicht nachzuvollziehen, dass man nicht einfach die Werte des exponentiellen und beschränkten Wachstums addieren kann. Das Programm (s. Abb. 47) berechnet beide Werte und bildet danach das geometrische Mittel.

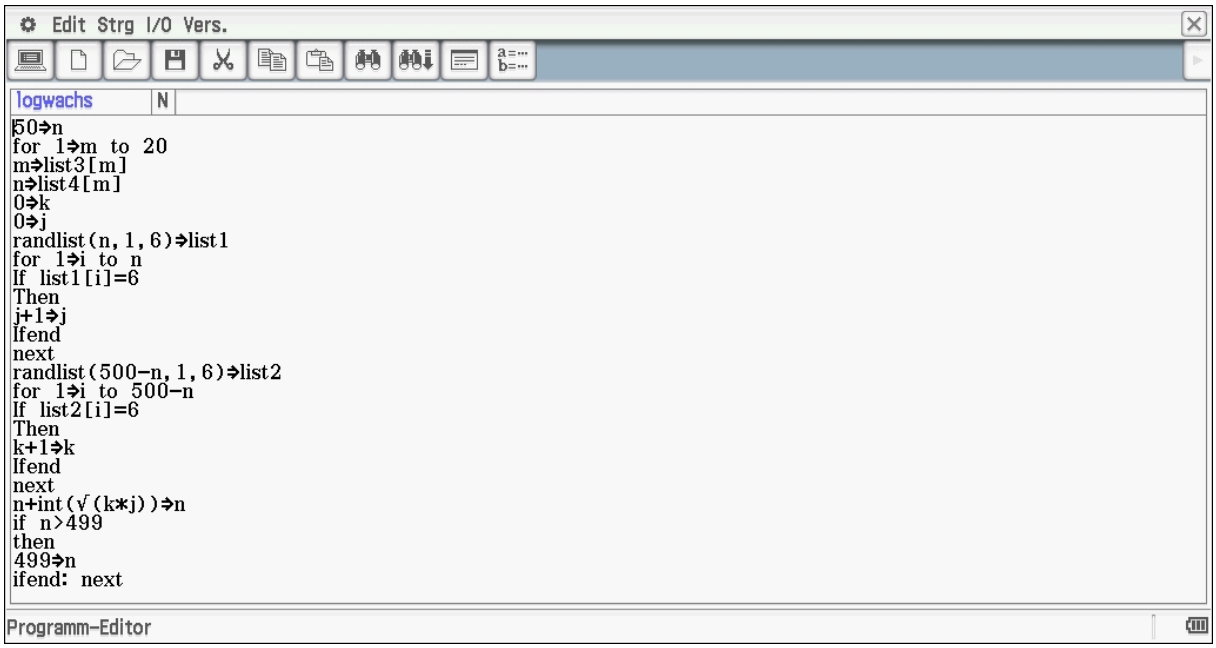


Abb. 47 Programm für das logistische Wachstum

Die Begrenzung $n>499$ (vorletzte Zeile links) musste eingeführt werden, da man ansonsten mit dem Befehl *randlist(500-n,1,6)* Probleme bekommt.

Die Ergebnisse sind in der folgenden Abbildung dargestellt.

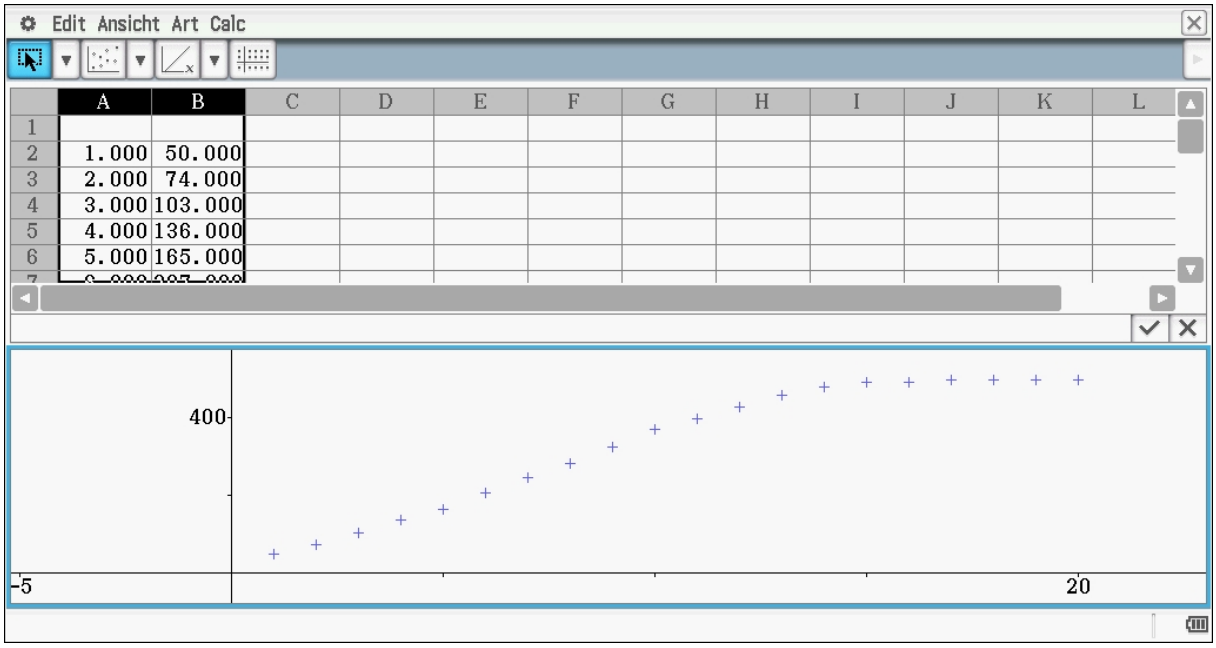


Abb. 48 Grafische Darstellung der Ergebnisse einer Simulation für das logistische Wachstum

5.3 Mathematische Experimente

Grafische Spielereien

Der ClassPad II bietet vielfältige Möglichkeiten, Kurven und Funktionsgraphen mit Hilfe des Geometrie- oder des Funktionsgraphen-Moduls darzustellen. Für Schülerinnen und Schüler kann es aber verständnisfördernd sein, wenn zum Beispiel eine Gerade durch einen wandernden Punkt erzeugt werden soll.

Das folgende Programm zeigt einen Punkt, der in Richtung der x-Achse wandert und der entstehende Strahl an dieser gespiegelt wird.

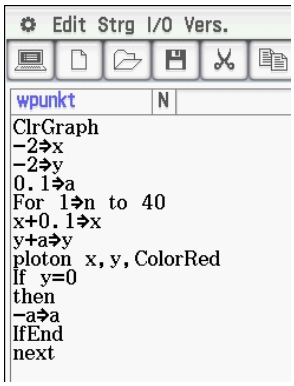


Abb. 49 Wandernder und gespiegelter Punkt

Wie die Abbildung 50 zeigt, liegt dem Befehl *ploton* automatisch das Koordinatensystem des Funktionsgrafen-Moduls zugrunde. Für die Ausführung des Programms ist es erforderlich, dass zunächst der relevante Bereich im Funktionsgrafenmodul festgelegt wird. Fügt man hinter dem Befehl *ploton x,y,ColorRed* den Befehl *plotoff x,y* ein, so wandert der Punkt und seine Bahn wird wieder gelöscht.

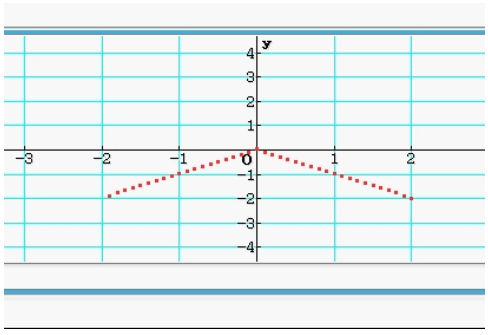


Abb. 50 Bahn des wandernden und gespiegelten Punktes

Das Ganze lässt sich erweitern, wenn man ein Rechteck vorgibt und in diesem den Punkt wandern lässt. Die folgende Abbildung zeigt das entsprechende Programm.

```
line 4,2,4,-2,ColorGreen
line 4,-2,-4,-2, ColorGreen
line -4,-2,-4,2, ColorGreen
line -4,2,4,2, ColorGreen
-4⇒x: 0⇒y
-4⇒a: 0⇒d
0.05⇒b
0.1⇒c
x+b⇒x: y+c⇒y
While (not(x=-4)) or (not(y=0))
ploton x,y, ColorRed
x+b⇒x
y+c⇒y
if (x=4) or (x=-4)
then
-b⇒b
ifend
if (y=2) or (y=-2)
then
-c⇒c
ifend
plotoff x,y
WhileEnd
```

Abb. 51 Programm für einen im Rechteck wandernden Punkt.

Das Programm ist so aufgebaut, dass es beendet wird, wenn der Ausgangspunkt wieder erreicht wird. Man kann natürlich jeden Programmablauf durch die Betätigung der *Esc*-Taste stoppen, was natürlich nicht besonders elegant ist. Mit Schülerinnen und Schülern ist es nun interessant zu diskutieren, wie das Verhältnis der Zuwächse (Zeilen 7 und 8) zu wählen ist, damit es zu einem Abbruch kommt. Des Weiteren kann man diskutieren, ob es jemals zu einem Abbruch kommt, wenn dies nicht beim ersten Mal geschieht, wenn x wieder den Wert -4 erreicht hat.

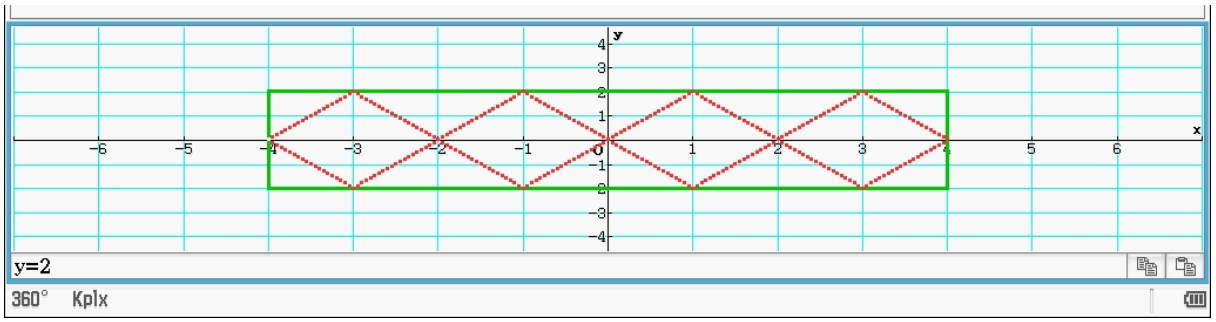


Abb. 52 Bahn des an einem Rechteck gespiegelten Punktes

Genau wie oben beschrieben, lässt sich durch Einfügen des Befehls *plotoff x,y* erreichen, dass nur ein wandernder Punkt aber nicht die Bahn dargestellt wird. Eine weitere Frage wäre, ob es durch die Wahl eines entsprechenden Verhältnisses möglich ist, jeden Punkt des Randes zu erreichen.

Spiel mit dem Zufall und Termverständnis[6]

Mit Hilfe des Befehls *rand()* lassen sich Zufallszahlen im Intervall (0,1) erzeugen. Lässt man diesen Befehl zweimal ausführen, so kann man die erhaltenen Werte als x- und y-Koordinaten eines Punktes verstehen, der sich dann bildlich darstellen lässt.

[6] Die Idee zu den Inhalten stammt aus [17]

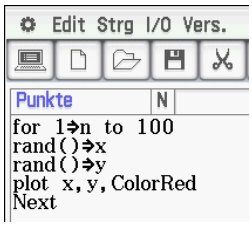


Abb. 53 Erzeugung von Zufallspunkten

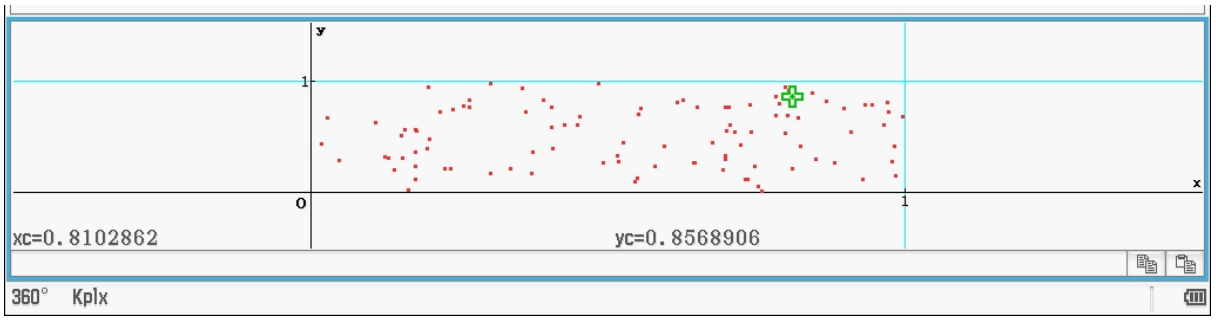


Abb. 54 Darstellung der erhaltenen Punkte

Durch die Darstellung bietet es sich an, über Verteilungen zu diskutieren. Dies soll hier allerdings nicht geschehen; sondern wir wollen die erhaltenen Punkte als Eckpunkte eines Rechtecks interpretieren und zusätzlich den Flächeninhalt und Umfang in Abhängigkeit der einen Rechteckseite darstellen.

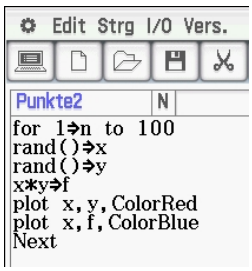


Abb. 55 Erzeugung von Zufallspunkten mit Flächeninhalt

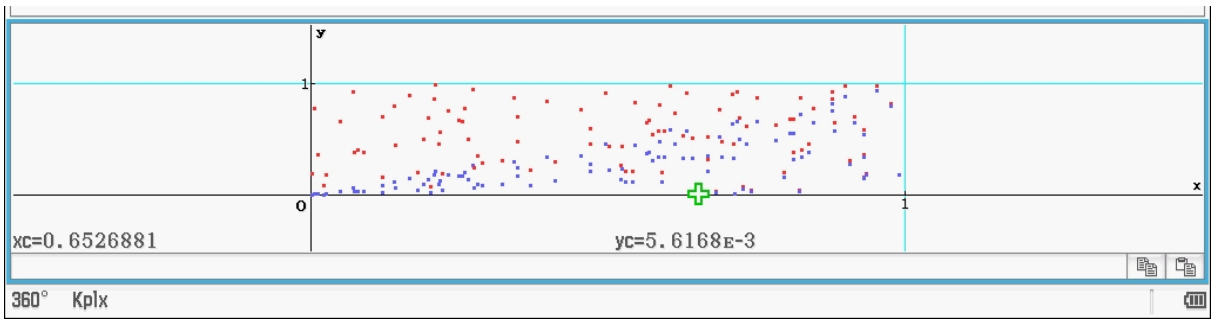


Abb. 56 Darstellung der Punkte und des Flächeninhaltes in Abhängigkeit der x-Koordinate

Man erkennt, dass die blauen Punkte, die den Flächeninhalt in Abhängigkeit von der *x*-Koordinate wiedergeben, offensichtlich in einem rechtwinkligen Dreieck mit der Geraden $f(x) = x$ als Hypotenuse liegen. Die beiden Katheten ergeben sich automatisch aus den Voraussetzungen. Da $x < 1$ und $y < 1$ folgt natürlich $x \cdot y < 1$. Die Begründung für

die Begrenzung durch die Winkelhalbierende ergibt sich auch direkt. Da $y < 1$ folgt $x \cdot y < 1$.

Als nächstes wollen wir uns dem Umfang widmen.

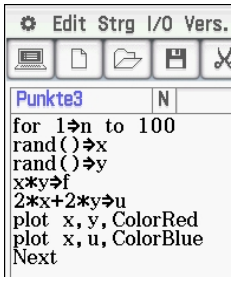


Abb. 57 Programm zur Darstellung des Umfangs in Abhängigkeit von der x-Koordinate

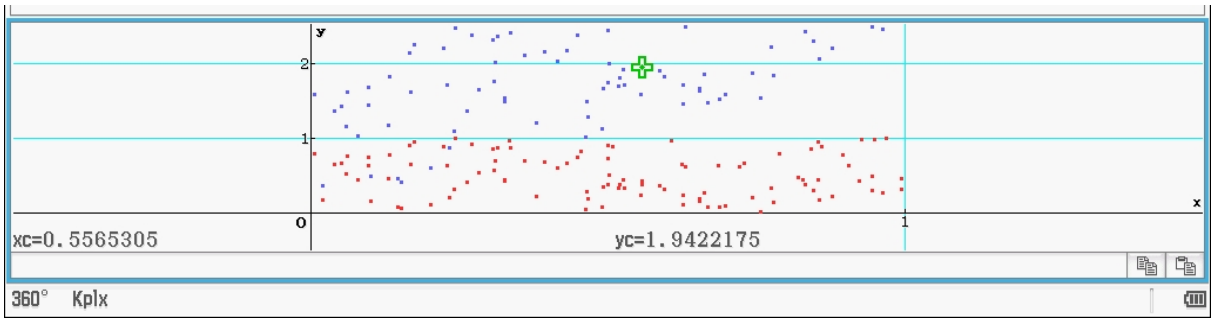


Abb. 58 Darstellung der Punkte und des Umfangs in Abhängigkeit der x-Koordinate

Aus der Darstellung ergibt sich zunächst nichts, was sich lohnen würde, genauer mathematisch analysiert zu werden. Um sich einen besseren Überblick zu verschaffen, wird die Anzahl der Punkte auf 200 erhöht und der Bereich in *y*-Richtung vergrößert.

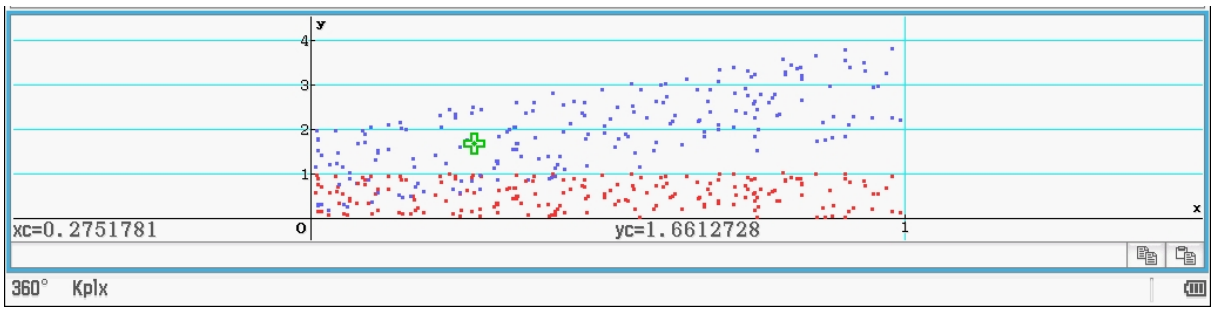


Abb. 59 Darstellung der 200 Punkte und des Umfangs in Abhängigkeit der x-Koordinate

Aus der Abbildung 59 ist jetzt deutlich eine Punktmenge erkennbar, die von vier Geraden begrenzt wird. Interessant sind vor allem die Geraden in *y*-Richtung; die beiden anderen ergeben sich automatisch aus der Begrenzung der *x*-Werte. Die Gerade, die die Menge in *y*-Richtung nach unten begrenzt, ergibt sich direkt aus $U = 2a + 2b$. Da $b > 0$, gilt $U > 2a$ und $g_1 : f(x) = 2x$. Mit einer ähnlichen Überlegung ergibt sich aus $b < 1$: $U < 2a + 2$ und $g_2 : f(x) = 2x + 2$. Interessanter ist es, die Abhängigkeit des Flächeninhalts vom Umfang darzustellen.

```
Edit Strg I/O Vers.
Punkte4 N
for 1➔n to 200
rand()➔x
rand()➔y
x*y➔f
2*x+2*y➔u
ploton u,f, ColorBlue
Next
```

Abb. 60 Programm zur Darstellung des Flächeninhaltes in Abhängigkeit vom Umfang

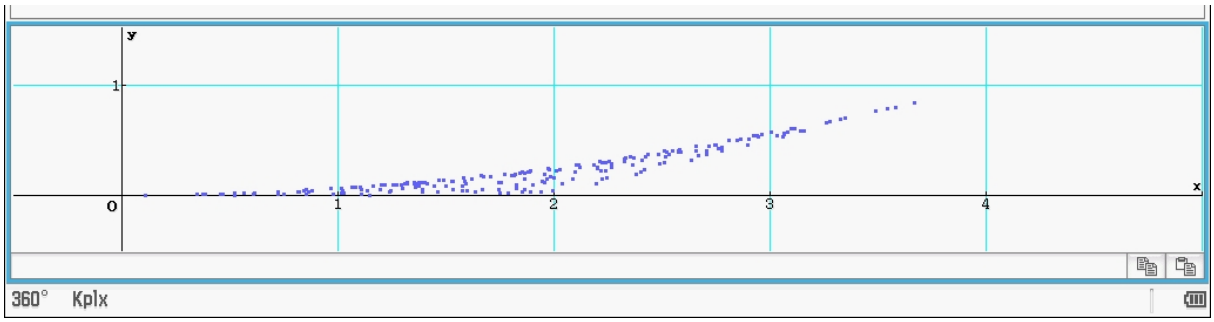


Abb. 61 Darstellung des Flächeninhalts in Abhängigkeit vom Umfang

Aus der Darstellung (Abb. 61) lässt sich vermuten, dass die Punkmenge nach oben durch eine Parabel und nach unten für $x > 2$ durch eine Gerade begrenzt ist. Zur Überprüfung und Bestätigung werden die entsprechenden Terme herangezogen.

Es gilt: $F = a \cdot b$ und $U = 2a + 2b$.

Wir lösen die zweite Gleichung nach *b* auf und setzen dies in die erste ein.

Es ergibt sich $F = a\left(\frac{U}{2} - a\right) = -a^2 + \frac{U}{2} \cdot a \qquad (*)$

Der Ausdruck (*) beschreibt eine nach unten geöffnete Parabel. Für die Abschätzung interessiert uns das Maximum. Die beiden Nullstellen haben die Werte $a_1 = 0$ und $a_2 = \frac{U}{2}$.

Das Maximum erhält man für $a_m = \frac{U}{4}$ und die maximale Fläche ergibt sich zu $F_m = \frac{U^2}{16}$.

Für Schülerinnen und Schüler ist es n dieser Stelle natürlich hilfreich den Graphen der Funktion mit in das obige Bild hinein zeichnen zu lassen. Dies ist aber leider nicht so einfach möglich. Wenn man nach Beendigung des Programms sich den Graphen einer zusätzlichen Funktion zeichnen lassen will, so wird vorher der Bildschirm gelöscht.

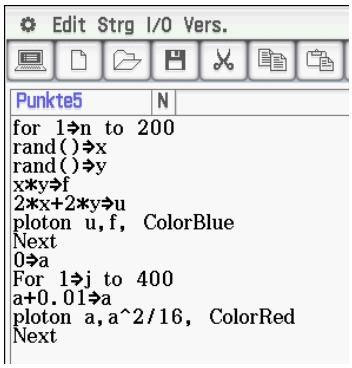


Abb. 62 Programm mit "oberer" Grenzfunktion

Die Abbildung 62 zeigt den Ausweg aus der oben geschilderten Problematik. Der Graph der Grenzfunktion wird mit Hilfe der zweiten *For*-Schleife punktweise geplottet. Das Ergebnis zeigt die folgende Abbildung 63.

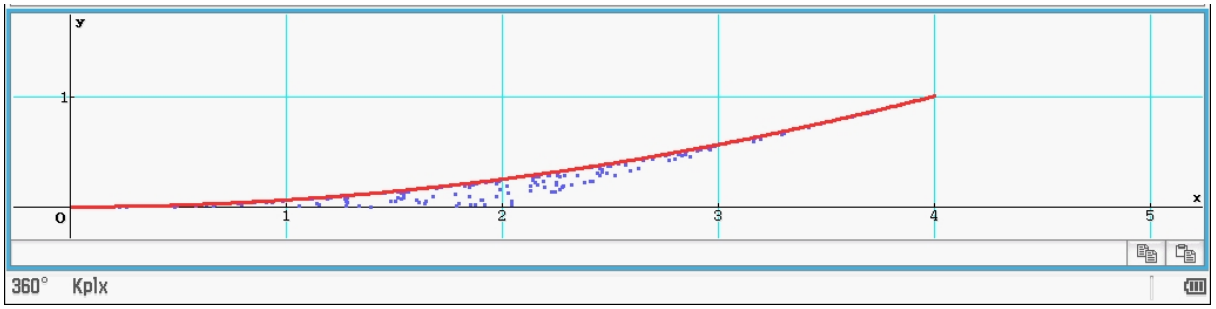


Abb. 63 Punktmenge mit Grenzfunktion

Es fehlt noch die untere Begrenzung. Es ist leicht einsehbar, dass dies für $U < 2$ die x-Achse ist, da aus $U < 2$ folgt: $a = 0$ oder $b = 0$ und damit $F = 0$.

Ist $U > 2$, dann kann weder $A = 0$ noch $b = 0$ gelten. Insbesondere muss dann $a + b > 1$ (*) gelten. Der Flächeninhalt wird minimal, wenn a oder b minimal sind. Wegen (*) ist dies der Fall, wenn $a = 1$ oder $b = 1$ gilt.

Es sei $b = 1$. Dann ist $F = a$ und $U = 2(a+1)$, also $F(U) = \frac{U}{2} - 1$ (**).

Durch (**) wird gerade die untere Gerade beschrieben (s. Abb. 64). Erzeugt wird das Bild, indem man in das Programm (s. Abb. 62) vor das letzte *Next* die folgende Zeile einfügt: *ploton a,a/2-1, ColorGreen*

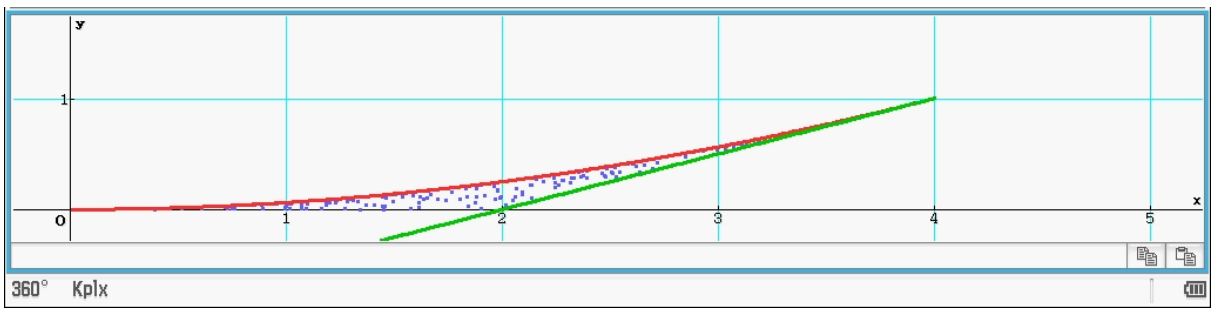


Abb. 64 Punktmenge mit beiden Grenzfunktionen

Bisher sind wir von Punkte ausgegangen, die in einem Quadrat eingesperrt waren. Man kann die obigen Fragestellungen in der Hinsicht verändern, dass man nun als Ausgangsmenge Punkte betrachtet, die in einem rechtwinkligen Dreieck liegen. Man erhält eine solche Menge zum Beispiel dadurch, dass man die Punkte wieder nach obigem Zufallsprinzip auswählt. Wir beschränken die Menge dadurch, dass wir $x+y<1$ fordern. Dies zu erreichen hat man die Möglichkeit, die Punkte, für die $x+y>1$ gilt, auszuschließen oder die Punkte an der Geraden $x+y=1$ zu spiegeln.

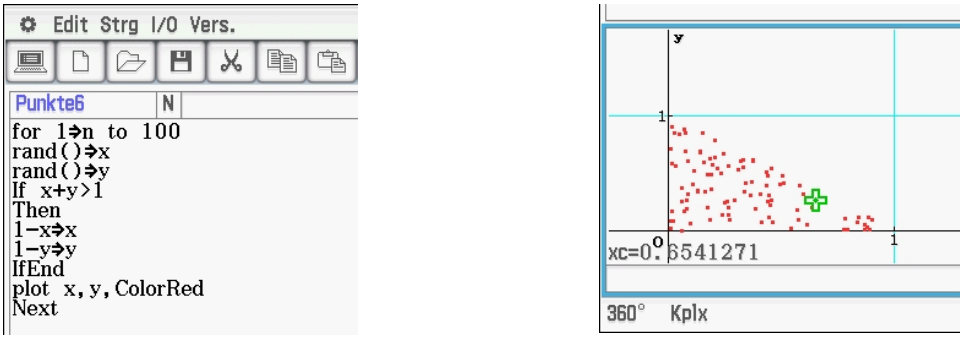


Abb. 65 In einem Dreieck eingesperrte Punkte

Man kann nun mit diesen Punkten ebenso verfahren wie oben und hat damit sehr viele Differenzierungsmöglichkeiten für den Unterricht. Des Weiteren sei darauf hingewiesen, dass sich die in diesem Abschnitt beschriebenen Erkundungen auch mit einer Tabellenkalkulation durchgeführt werden können.

Das Newton-Verfahren für komplexe Funktionen

Das Newton-Verfahren lässt sich auch auf komplexe Funktionen anwenden. Wir betrachten z.B. die Funktion $f(z)=z^3-1$. Diese Funktion hat bekanntlich drei Nullstellen. Interessant ist nun die Frage, gegen welche der drei Nullstellen die Folge konvergiert, wenn man den Startwert verändert. Dazu wurde der Bereich $-1{,}5 \le \mathrm{Re}(z) < 1{,}5$ und $-1{,}5 \le \mathrm{Im}(z) < 1{,}5$ untersucht. Die Schrittweite betrug 0,1, und die Berechnungen werden 5 mal durchgeführt. Das Programm ist im Folgenden wiedergegeben (s. Abb. 66).

```
Edit Strg I/O Vers.
new1 N
-1.5⇒x
for 1⇒k to 30
1.5⇒y
for 1⇒l to 30
x⇒a
y⇒b
for 1⇒n to 5
a⇒h
2/3*a+1/3*(a^2-b^2)/(a^2+b^2)^2⇒a
2/3*(b-h*b/(h^2+b^2)^2)⇒b
Next
if (1-a)^2+b^2<0.1
then
plot x,y,ColorBlue
ifend
if (-cos(60)-a)^2+(sin(60)-b)^2<0.1
then
plot x,y,ColorRed
ifend
if (-cos(60)-a)^2+(-sin(60)-b)^2<0.1
then
plot x,y,ColorGreen
ifend
y-0.1⇒y
next
x+0.1⇒x
next
Programm-Editor
```

Abb. 66 Das Newton-Verfahren für komplexe Funktionen

Wie aus dem Programm hervorgeht wird die jeweilige Konvergenz farblich gekennzeichnet. Aus der Abbildung 67 geht hervor, dass eine relativ große Schrittweite gewählt wurde. Dies lässt sich natürlich verfeinern. Zu bedenken ist, dass der Rechenaufwand relativ groß ist. Dies gilt auch, wenn man nicht das Handheld sondern die Rechnerversion benutzt.

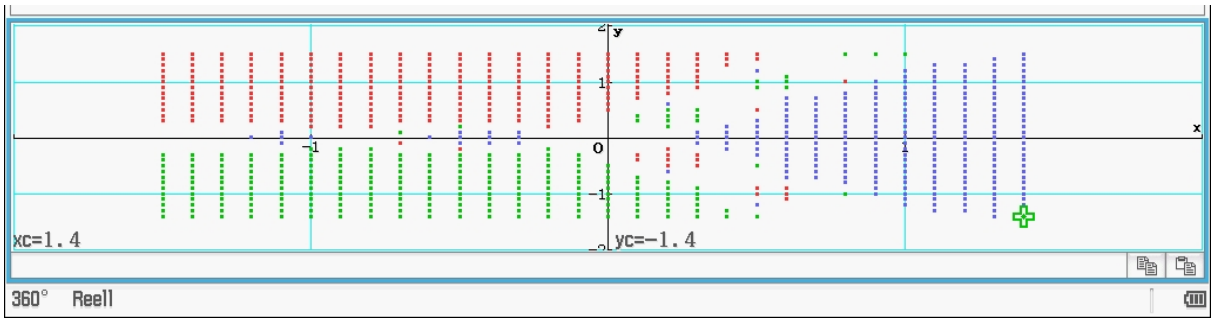


Abb. 67 Konvergenz des Newton-Verfahrens für f(z)=z^3-1

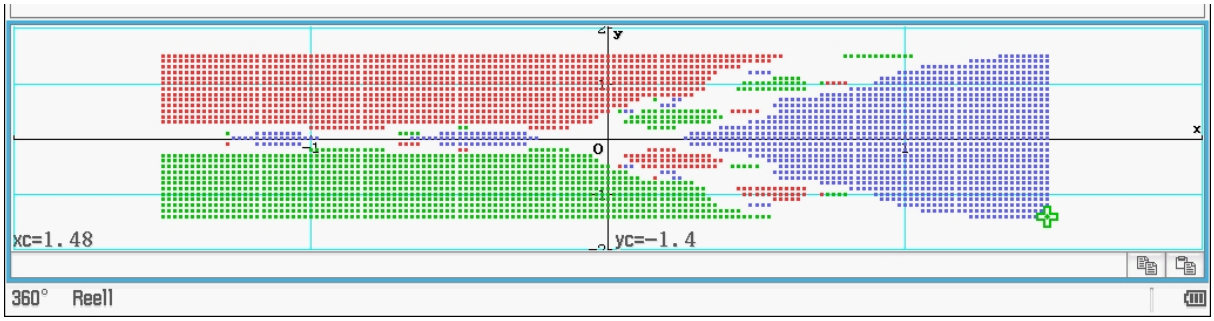


Abb. 68 Bild mit kleinerer Schrittweite

Man erkennt deutlich, dass es sich bei der gekennzeichneten Menge offensichtlich nicht um eine zusammenhängende Menge handelt. Es gibt mehrere singuläre Bereiche. Des Weiteren hat die Menge eine fraktale Struktur. Die Erzeugung des Bildes (Abb. 68) dauert auf einem Netbook fast zwei Stunden.

Das Apfelmännchen

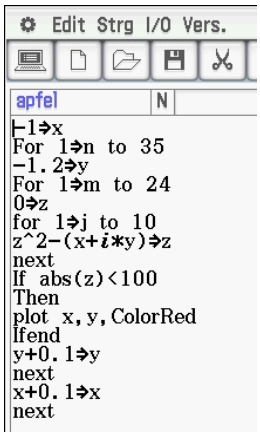


Abb. 69 Programm zur Erzeugung des Apfelmännchens

Die Erzeugung des Apfelmännchens durch B. Mandelbrodt war der Beginn der fraktalen Epoche. Es gibt im Netz viele Darstellungen verbunden mit der Möglichkeit, vor allem Teile des Randes beliebig zu vergrößern. Die Abbildung 70 kann natürlich nicht an diese heranreichen. Immerhin ist es aber mit dem ClassPad II möglich, die grobe Struktur darzustellen. Für Schülerinnen und Schüler macht es schon einen Unterschied, ob man sich fertige Bilder anschaut oder am Entstehungsprozess beteiligt ist.

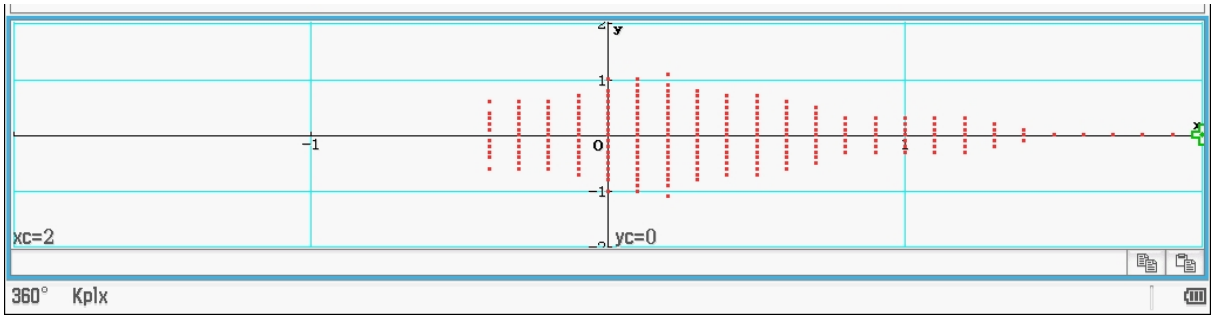


Abb. 70 Grobe Struktur des Apfelmännchens

Wenn man das Bild genauer haben möchte, muss man erheblich mehr Zeit investieren.

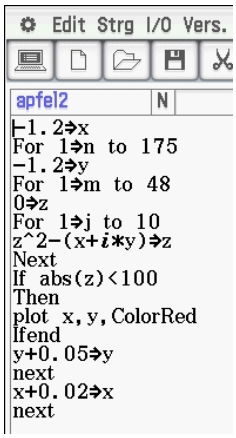


Abb. 71 Programm zur Erzeugung des Apfelmännchens mit einer feineren Struktur

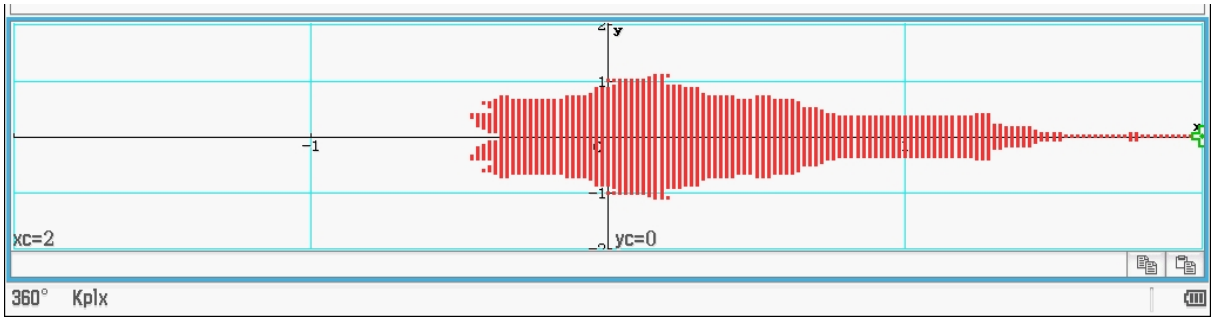


Abb. 72 Das Apfelmännchen (etwas sehr gestreckt)

Der Random-Walk

Simulationen eines Random-Walks sind vor allem für die Physik bedeutsam. Der zweite Hauptsatz der Thermodynamik sagt aus, dass die Entropie zunimmt. Geht man zum Beispiel von einem Gefäß aus, indem sich z.B. ein Gas befindet, so ist es unmöglich von einem Endzustand auf einen Anfangszustand zu schließen. Eine Simulation mit hinreichend vielen Teilchen geht über die Möglichkeiten des ClassPads II hinaus. Es ist aber möglich, einen Random-Walk für ein Teilchen zu simulieren. Dazu geht man z.B. von 100 Schritten aus und nach jedem Schritt bewegt sich das Teilchen mit einer Wahrscheinlichkeit von 50% nach links oder nach rechts. Das folgende Programm zeigt die Simulation von 100 solcher Walks.

```
walk1                    N
For 1⇒n to 100
0⇒x
For 1⇒m to 100
rand(0,1)⇒w
If w=1
then
x+1⇒x
else
x−1⇒x
IfEnd
next
n⇒list1[n]
x⇒list2[n]
next
For 1⇒n to 100
list3[n]=0
next
For 1⇒m to 50
0⇒z
For 1⇒n to 100
If abs(list2[n])=m
then
z+1⇒z
IfEnd
next
z⇒list3[m]
next
```

Abb. 73 Programm zur Erzeugung von Daten für einen Random Walk

Um die verschiedenen Ausgänge statistisch zu erfassen, werden im Programm „walk1“ die Werte in der Liste „list2“ gespeichert. Mit Hilfe des Statistik-Moduls lassen sich die Häufigkeiten dann veranschaulichen. Als Startwert wurde „0“ benutzt. Dieser lässt sich natürlich genauso verändern, wie die Anzahl der Schritte und die Anzahl der Simulationen. Alternativ kann man auch die Anzahlen direkt im Programm erzeugen. Dies geschieht mit den beiden letzten *For*-Schleifen. Für eine Auswertung mit dem Statistik-Modul kann man diesen Teil dann natürlich streichen.

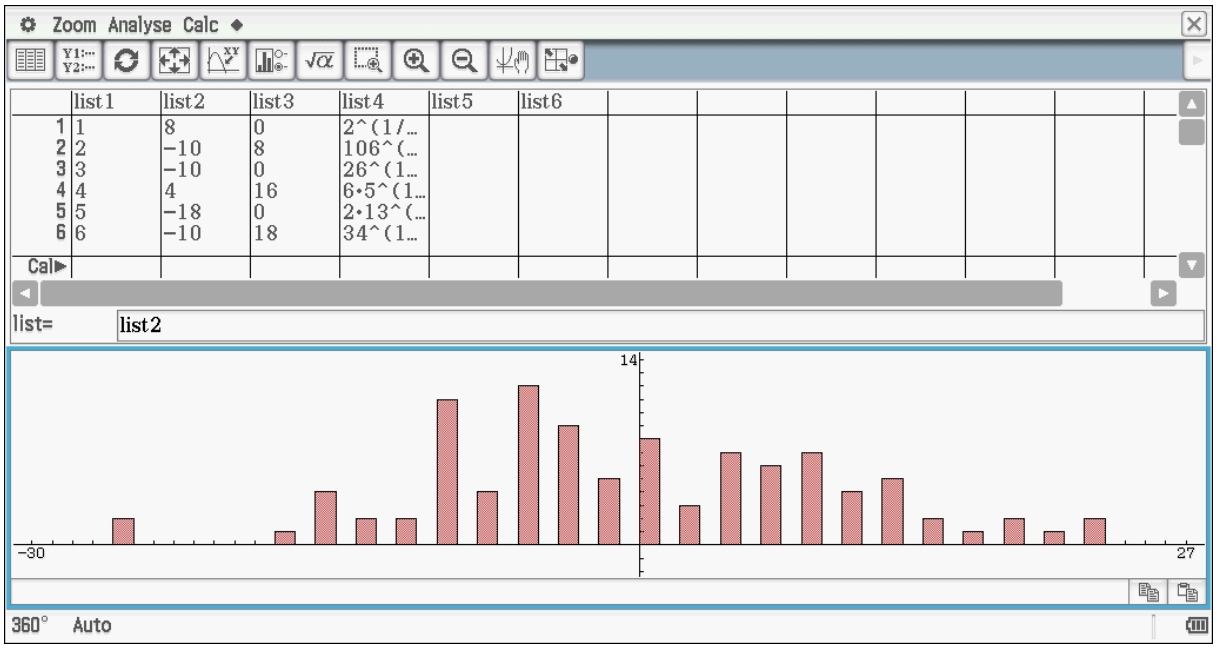


Abb. 74 Darstellung der Werte eines Random Walks

Für einen Vergleich der Werte mit dem zwei- oder dreidimensionalen Fall ist eine Verteilung bezüglich des Abstands hilfreich. Dazu erzeugt man sich eine neue Liste „list3“ durch *abs(list2)=>list3* im *main*-Bereich.

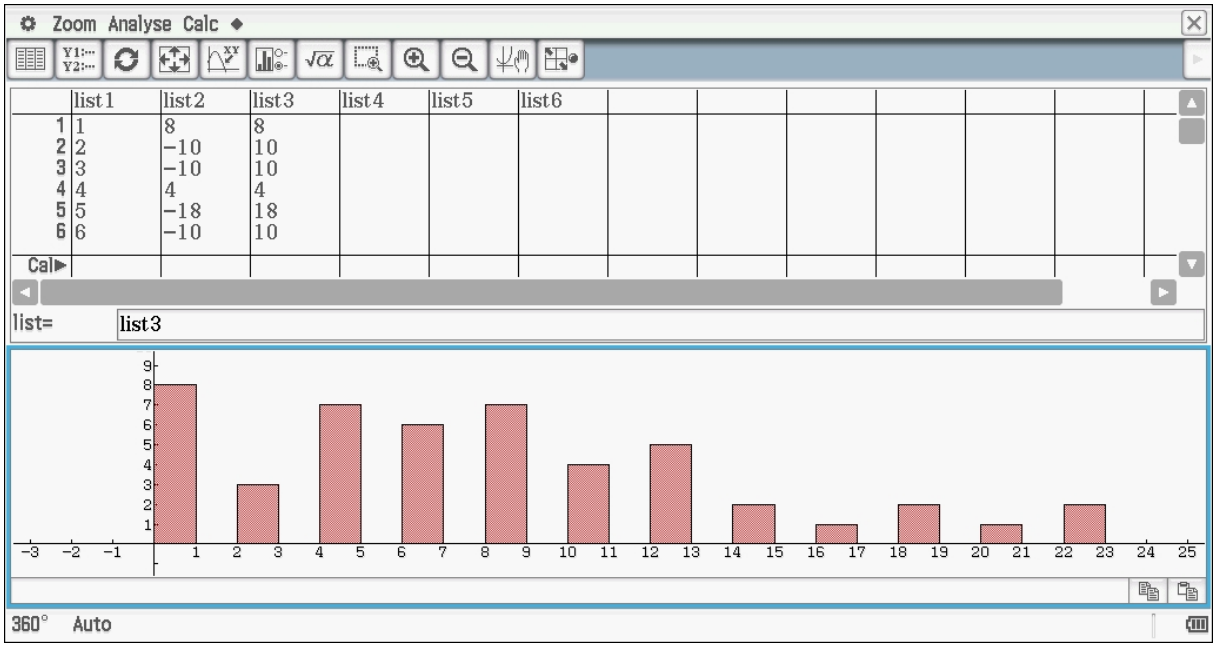


Abb. 75 Verteilung der Abstände vom Koordinatenursprung

Der zweidimensionale Fall

Das folgende Programm dient dazu, die erforderlichen Werte zu erzeugen.

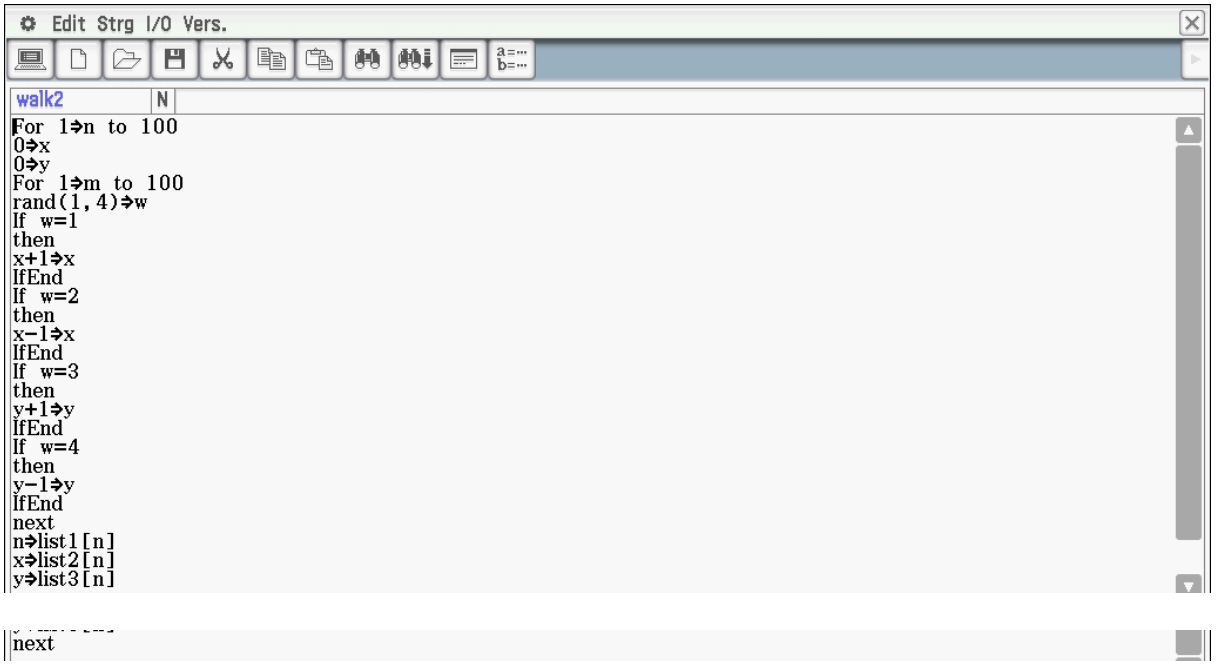


Abb. 76 Programm zur Simulation eines zweidimensionalen Random-Walks

Zur Auswertung sind noch aus den Listen list2 und list3 die Abstände zu berechnen. Da natürlich das Statistik-Programm nur ganzzahlige Werte in einem Balkendiagramm darstellen kann, müssen die Werte gerundet werden. Dies geschieht im main-Bereich

mit der Funktion *int*. Für eine korrekte Rundung muss man die Werte noch um 0,5 vergrößern. Die folgende Abbildung zeigt die sich ergebende Verteilung. Man erkennt einen deutlichen Unterschied zum eindimensionalen Fall. Es ergibt sich eine gewisse Ähnlichkeit zur Normalverteilung, die im eindimensionalen Fall überhaupt nicht gegeben ist. Eine theoretische Auswertung der Experimente kann im Schulunterricht sicher nur angedeutet werden.

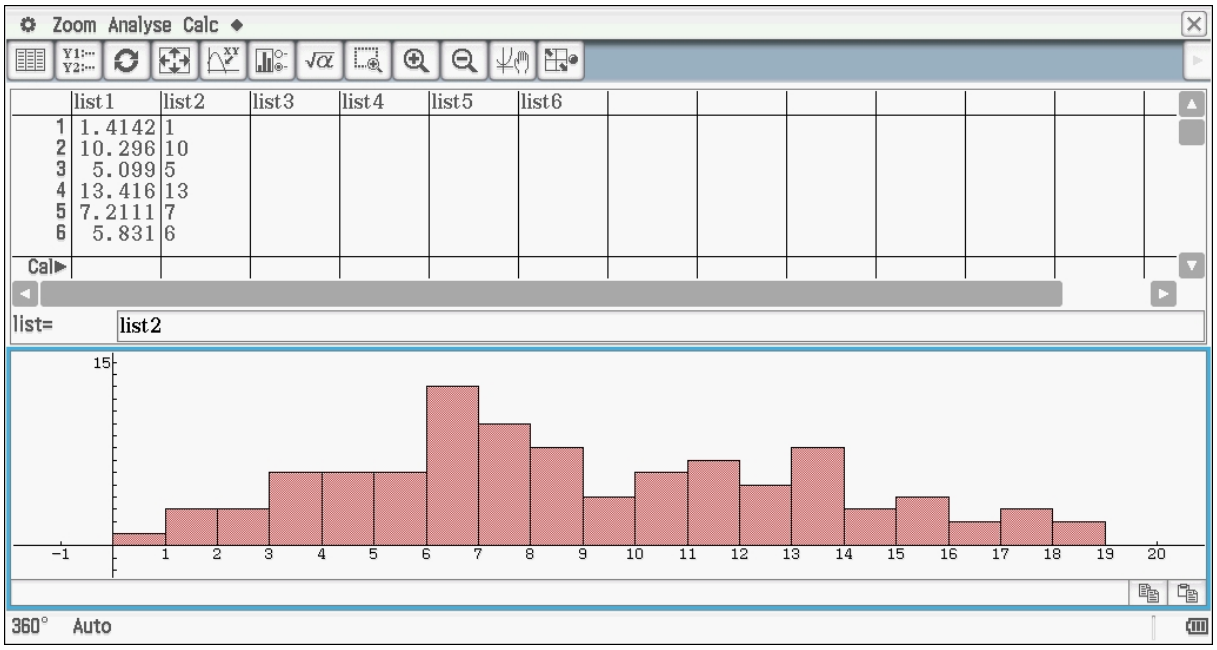


Abb. 77 Verteilung der Abstände für den 2-dimensionalen Random Walk

5.4 Programme zur Simulation stochastischer Prozesse

Das Münzenproblem

Man hat drei Münzen, eine „normale“ Münze mit Kopf und Zahl, eine mit zweimal Kopf und eine mit zweimal Zahl. Es wird zufällig eine der Münzen ausgewählt und geworfen. Die geworfene Münze zeigt „Kopf“. Wie groß ist die Wahrscheinlichkeit, dass auf der anderen Seite „Zahl“ liegt? Da nur zwei Münzen in Frage kommen, sollte man meinen, dass diese Wahrscheinlichkeit 50% beträgt. Mit dem folgenden Programm lassen sich Simulationen durchführen, die ein anderes Ergebnis zeigen.

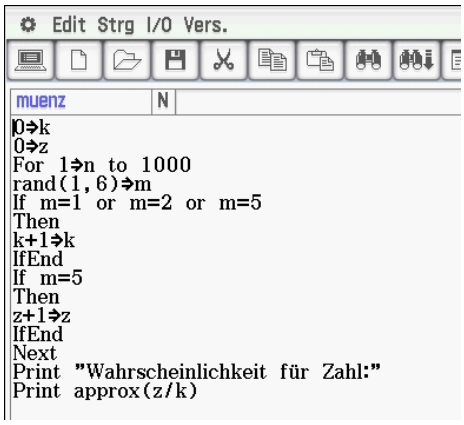


Abb. 78 Programm zur Simulation des Münzenproblems

Aus dem Programm wird deutlich, dass zunächst alle möglichen 6 Fälle zu betrachten sind. In drei Fällen (1, 2 oder 5) kann der Fall eintreten, dass „Kopf“ oben liegt. Die Zahl liegt nur dann unten, wenn die gemischte Münze ($m = 5$) gefallen ist.

Als Rechnung erhält man:

$$P(\text{Zahl unten} \mid \text{Kopf oben}) = \frac{P(\text{Zahl unten und Kopf oben})}{P(\text{Kopf oben})} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}.$$

Das Ziegenproblem

In einer Fernsehsendung kann ein Kandidat zwischen drei verschiedenen Toren wählen. Hinter zwei Toren ist eine *Ziege* und hinter einem Tor ein *Auto* verborgen. Nachdem der Kandidat sich für ein Tor entschieden hat, öffnet der Moderator ein Tor, hinter dem natürlich eine *Ziege* steht. Die Frage ist nun, ob der Kandidat seine erste Entscheidung ändert oder nicht. Vor Jahren gab es über die Lösung des Problems sogar eine Diskussion bis in die mathematischen Fakultäten der Universitäten. Auch dieses Problem lässt sich mit Hilfe des ClassPad II simulieren. Im Folgenden ist ein Programm wieder gegeben, mit dessen Hilfe sich der Prozess simulieren lässt.

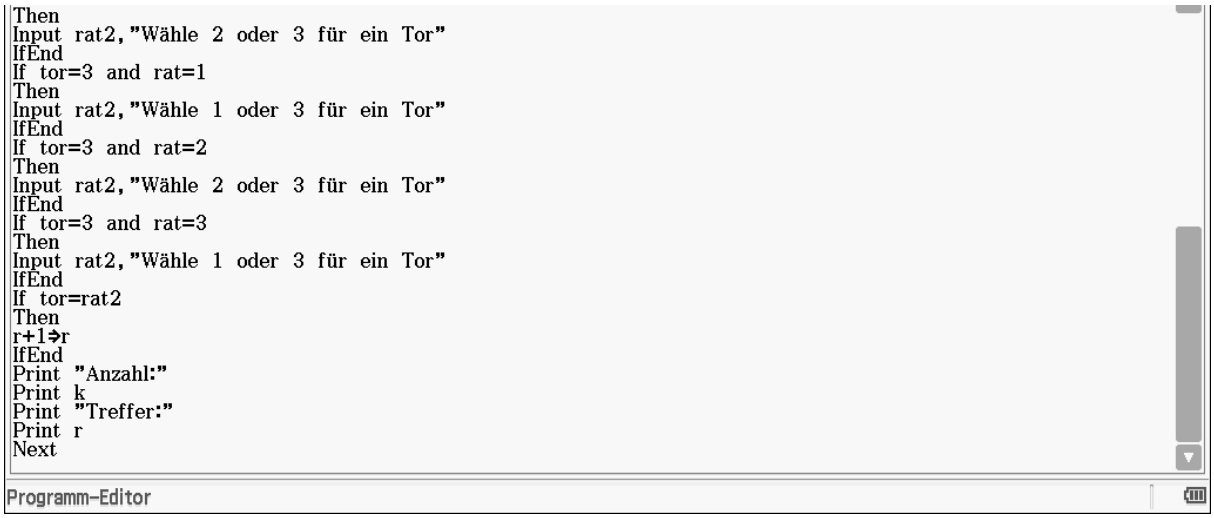


Abb. 79 Programm zur Simulation des Ziegenproblems

Um das Programm möglichst einfach und übersichtlich zu halten, wurde darauf verzichtet, die Fälle, dass der Spieler andere Zahlen als 1, 2 oder 3 wählt, auszuschließen. Entscheidend ist, dass durch das Erstellen des Programms zu erkennen ist, dass der Kandidat nur in dem Fall, dass er zu Anfang mit seinem Tipp richtig gelegen ist, durch den Wechsel des Tors einen Nachteil hat. In den beiden anderen möglichen Fällen hat er einen Vorteil. Schon daraus ergibt sich der Vorteil für den Wechsel. Dieser wird empirisch durch mehrmaliges Spielen bestätigt.

Das Schlüsselproblem

Die Flurkommode der Familie Such hat 8 Schubladen. Herr Such packt den Haustürschlüssel meistens in eine der Schubladen. Langjährige Untersuchungen haben ergeben, dass der Schlüssel mit einer Wahrscheinlichkeit von 80% in einer der Schubladen liegt. Frau Such hat nun schon 7 der 8 Schubladen geöffnet und den Schlüssel nicht gefunden. Wie groß ist die Wahrscheinlichkeit, dass der Schlüssel in der letzten Schublade liegt. Wir vermuten, dass diese ebenfalls 80% beträgt. Die folgende Simulation belehrt uns eines Besseren. Man erhält für diese Wahrscheinlichkeit den Wert $\frac{1}{3}$.

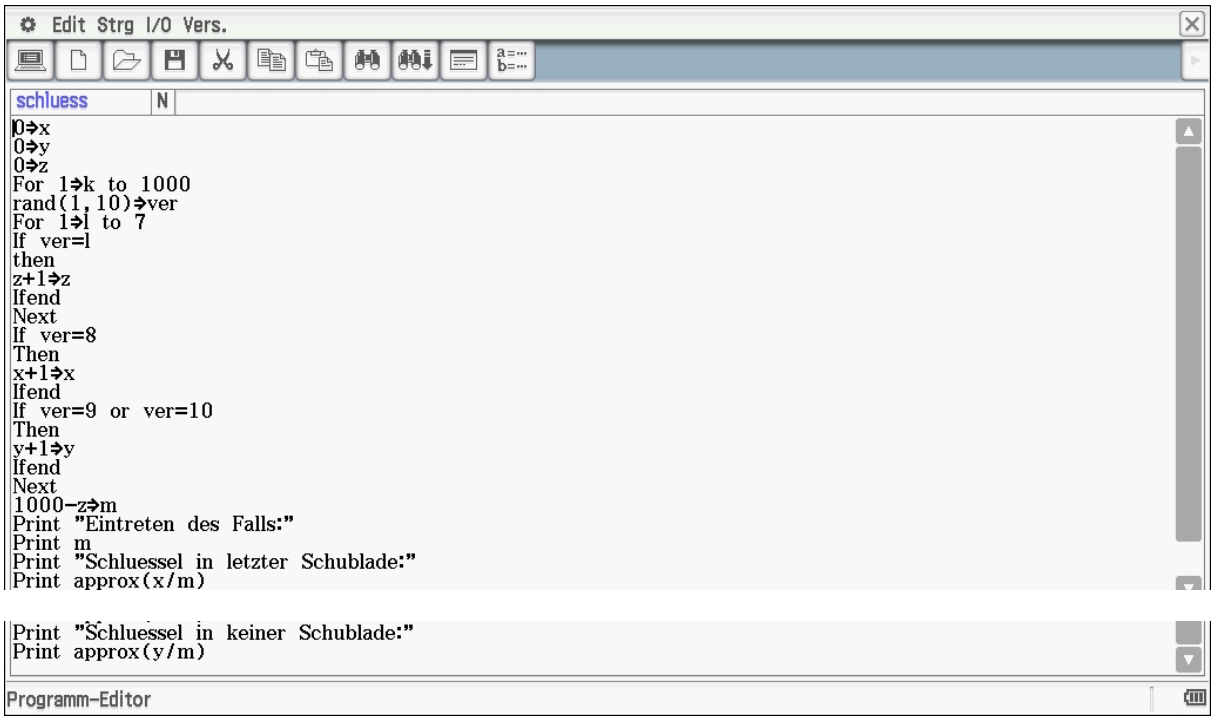


Abb. 80 Programm zur Simulation des Schlüsselproblems

Das Programm zeigt den Fehler, der gemacht wird, wenn man meint, dass die Wahrscheinlichkeit nach wie vor 80% beträgt. In ca. 70% der Fälle wird Frau Such den Schlüssel bereits in einer der ersten 7 Schubladen finden. Ein solcher Fall muss aber ausgeschlossen werden, da dieses ja schon überprüft wurde. Das heißt, nur in 30% der Fälle tritt überhaupt nur der Fall auf, dass sich der Schlüssel in der letzten Schublade oder woanders befindet.

Mit Hilfe von bedingten Wahrscheinlichkeiten lässt sich die Aufgabe folgendermaßen lösen:

Es bezeichne F das Ereignis „Der Haustürschlüssel befindet sich in der Flurkommode." und mit A sei das Ereignis „Der Haustürschlüssel befindet sich in der achten Schublade." bezeichnet. Mit $\overline{F}$ sei das Gegenereignis vom Ereignis F gemeint.

In der Aufgabe ist nach der bedingten Wahrscheinlichkeit $P(A \mid \overline{F} \cup A)$ gefragt.

Bekannt sind $P(F) = 0{,}8$ und $P(A \mid F) = \frac{1}{8}$.

Somit folgt aus der Definition für die bedingte Wahrscheinlichkeit

$$P(A \mid F) = \frac{P(A \cap F)}{P(F)} = \frac{P(A)}{P(F)} = \frac{P(A)}{0{,}8} = \frac{1}{8}, \text{ also } P(A) = 0{,}8 \cdot \frac{1}{8}.$$

Aus der Definition für die bedingte Wahrscheinlichkeit ergibt sich weiterhin

$$P(A \mid \overline{F} \cup A) = \frac{P(A \cap (\overline{F} \cup A))}{P(\overline{F} \cup A)} = \frac{P((A \cap \overline{F}) \cup (A \cap A))}{P(\overline{F}) + P(A) - P(\overline{F} \cap A)} =$$

$$= \frac{P(A \cup A)}{P(\overline{F}) + P(A) - P(\varnothing)} = \frac{P(A)}{P(\overline{F}) + P(A)} = \frac{0{,}8 \cdot \frac{1}{8}}{0{,}2 + 0{,}8 \cdot \frac{1}{8}} = \frac{1}{3}.$$

Das Problem Traummann

Ein Sprichwort sagt, dass frau nicht den Erstbesten heiraten soll. Die Frage, die sich nun aber stellt, ist die folgende, wann soll frau zugreifen. Um sich dem Problem zu nähern, wird zunächst angenommen, dass sich im Leben zehn Männer anbieten. Geht frau weiter davon aus, dass eine bestimmte Anzahl von Kandidaten abgelehnt wird, so lässt sich das Problem simulieren. Im Folgenden wird davon ausgegangen, dass die ersten 5 Kandidaten abgelehnt werden und nur dann ein Kandidat zum Zuge kommt, wenn er besser als die vorherigen Aspiranten ist.

Für eine solche Simulation muss zunächst eine zufällige Liste für in Frage kommende Kandidaten erzeugt werden. Dies geschieht, indem eine Liste mit 10 Zufallszahlen erzeugt wird. Diese Zahlen symbolisieren die Qualität der Kandidaten, das heißt, die größte Zufallszahl steht für den Traummann, die zweitgrößte für die zweite Wahl und so weiter. Dadurch ergibt sich eine Reihenfolge.

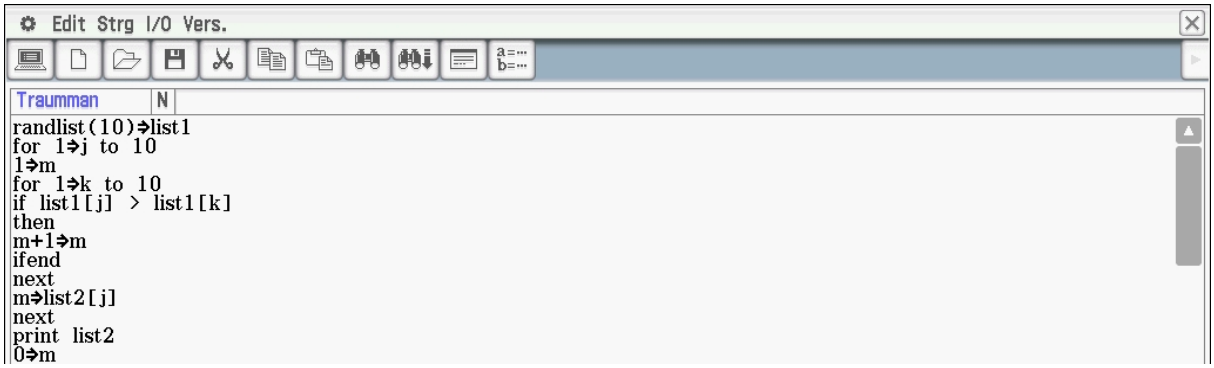


Abb. 81 Erzeugung einer zufälligen Anordnung der Zahlen 1 bis 10

In der List2 steht eine zufällige Anordnung der Zahlen von 1 bis 10. Es wurde oben schon gesagt, dass 5 Anträge erstmal nicht angenommen werden. Von den folgenden Kandidaten wird frau nur dann jemanden erwählen, wenn dieser besser als die ersten 5 ist. Daraus ergibt sich natürlich auch die Möglichkeit, dass frau den Rest ihres Lebens alleine verbringt.

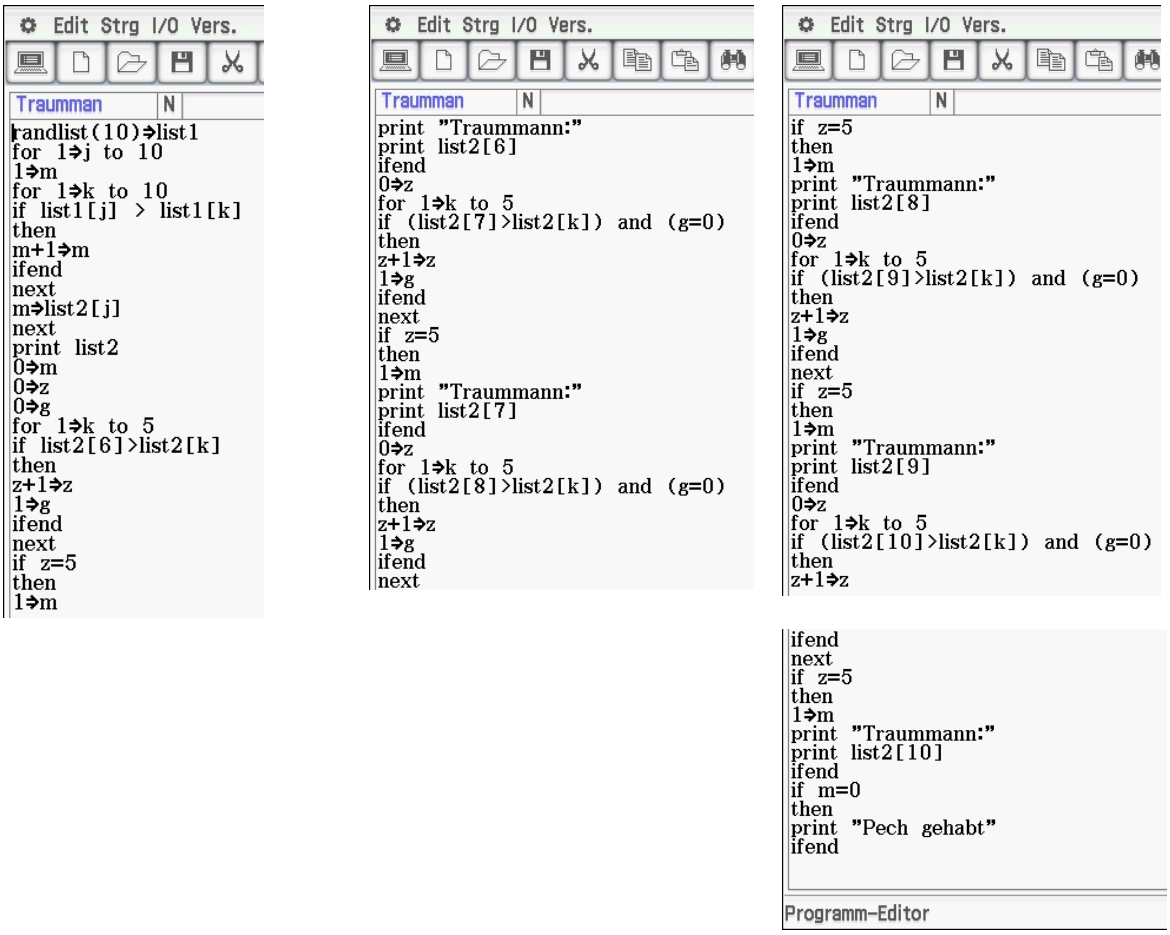


Abb. 82 Simulation für das Traummann Problem

Die Abbildung 82 zeigt das Programm für eine Simulation. Um zu einer Verteilung zu gelangen, ist die Simulation mehrfach auszuführen.

Wir lassen die Ergebnisse natürlich nicht mehr ausdrucken, sondern zählen, wie oft es zu keiner Heirat kam, wie oft der beste, der zweitbeste usw. gewählt wurde. Um dieses zu erreichen schließt man obiges Programm in einer weiteren „For-Schleife“ ein und legt eine „0“, eine „10“ für den besten, eine „9“ für den zweitbesten Ehemann in einer weiteren Liste list3 ab. Das Ergebnis lässt sich dann im Statistik-Modul veranschaulichen. Die folgende Abbildung zeigt einen möglichen Ausgang für 100 Simulationen.

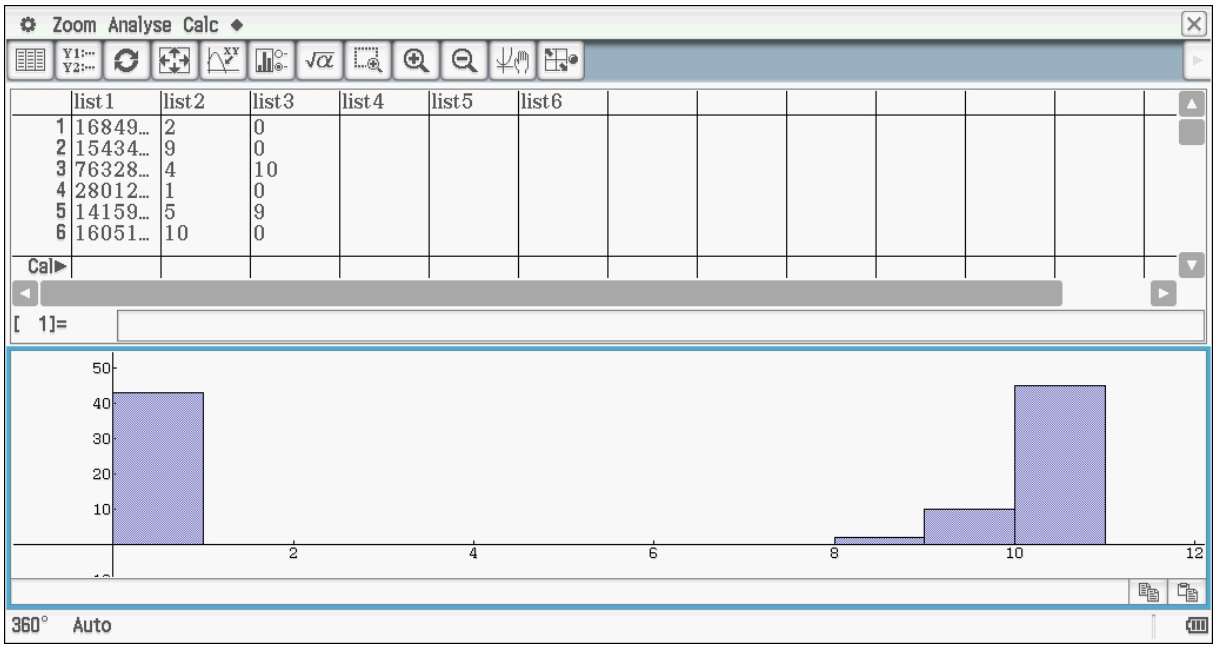


Abb. 83 Ergebnis von 100 Simulationen

Bevor man eine solche Abbildung erhält, muss die Grafik entsprechend der folgenden Abbildung eingestellt werden. Man gelangt in dieses Fenster, indem man in der oberen Symbolleiste das 4. Symbol von links wählt.

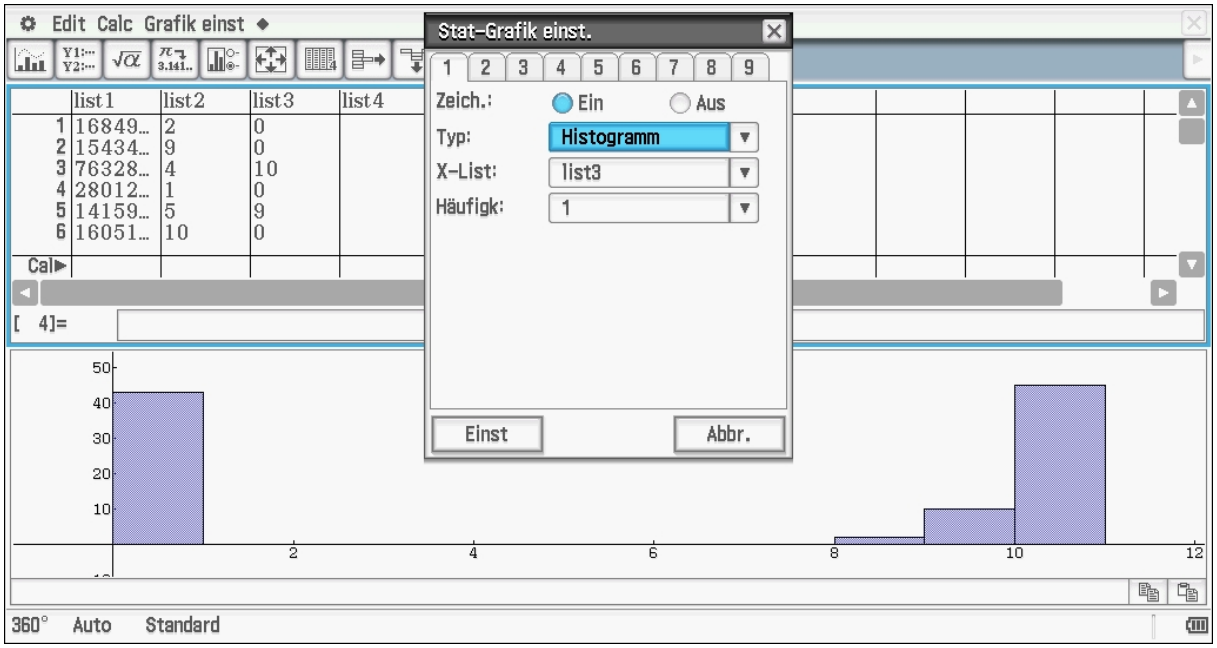


Abb. 84 Einstellung des ClassPad II

Aus der Abbildung wird deutlich, dass im Prinzip nur drei Fälle relevant sind:

1. frau bleibt ledig
2. frau bekommt den zweitbesten Kandidaten
3. frau bekommt ihren Traummann.

Das Problem lässt sich natürlich auch theoretisch analysieren. Wir wollen an dieser Stelle darauf verzichten und verweisen auf [15] oder [16].

Die Fußball Bundesliga

Man kann sich durchaus die Frage stellen, welche Rolle der Zufall bezüglich des Ausgangs eines Fußballspiels einnimmt. Oder anders gesprochen, lässt sich der Ausgang einer Budesligasaison voraussagen? Wir machen dazu folgende Annahmen:

1. Ein Viertel der Spiele endet unentschieden.
2. Für beide Mannschaften besteht die gleiche Gewinnchance.

Vor allem die zweite Annahme erscheint bei der Übermacht eines Münchener Vereins nicht sehr realistisch zu sein. Bezüglich solcher Modellierungen ist es aber zunächst empfehlenswert, sehr einfache Annahmen zu machen. Diese kann man dann später immer noch präzisieren. So findet man im Internet Abschlusstabellen vergangener Saisons, mit deren Hilfe man den Vereinen ein „Gewicht“ zuordnen könnte, um so Wahrscheinlichkeiten anders zu gewichten. Wie gesagt, wir gehen von den beiden obigen Annahmen aus und verzichten auf Unterscheidung der Vereine durch reale Namen; sie werden durch 1, 2, 3, …, 18 unterschieden.

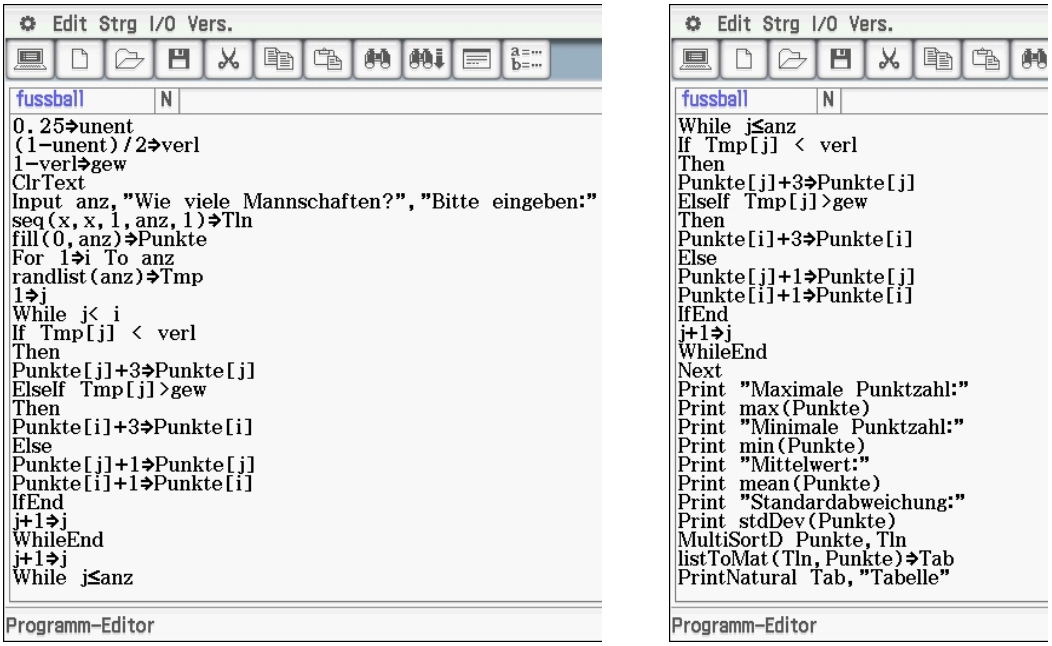


Abb. 85 Programm zur Simulation einer Bundesligasaison

Das entsprechende Programm ist in Abbildung 85 wiedergegeben. Zur leichteren Lesbarkeit wird die letzte Zeile unten links oben rechts wiederholt. Als Ergebnis erhält man zum Beispiel eine Abschlusstabelle, wie sie in Abbildung 86 wiedergegeben ist. Die Abschlusstabelle ist von links nach rechts zu lesen. Ein Vergleich mit den Tabellen der letzten Jahre zeigt, dass vor allem die Vereine, die in der Tabelle oben standen deutlich mehr Punkte hatten; das heißt, es sind die oben angesprochenen Korrekturen vorzunehmen.

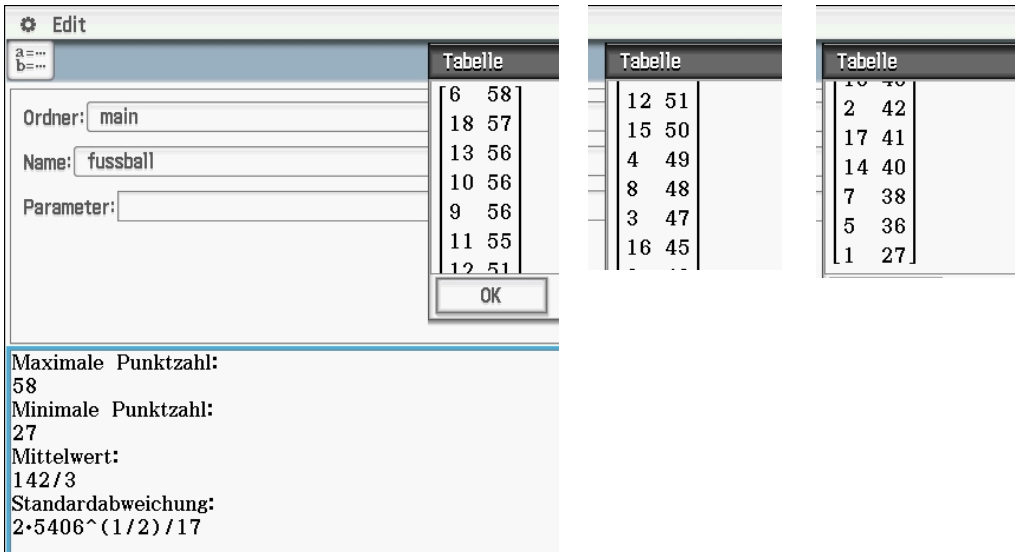


Abb. 86 Ergebnis einer simulierten Saison

6 Literatur und Quellen

[Handbuch] ClassPad II fx-CP400 Bedienungsanleitung, http://edu.casio.com

[1] Jonathan Borwein, Keith Devlin: Experimentelle Mathematik, Eine beispielorientierte Einführung, Spektrum, Heidelberg 2011, ISBN 978-3-8274-2661-1

[2] Christian Hesse: Das kleine Einmaleins des klaren Denkens, Becksche Reihe 2009, ISBN 978-3-406-58684-2

[3] ClassPad 330, ClassPad Betriebssystem, Version 3.06, Software Bedienungsanleitung
http://edu.casio.com
http://edu.casio.com/products/classpad/
http://edu.casio.com/dl/

[4] ClassPad 101 Lessons Materials
http://edu.casio.com/support/classpad101/lessonmaterials.html

[5] Udo Mühlfeld, Hannes Stoppel: Einführung in die Programmierung des ClassPad, Klassenstufe 11-13,

http://www.casio-schulrechner.de/de/download/materialdatenbank/Muehlenfeld_Einfuehrung_in_dieProgrammierung_desClassPad.pdf

[6] Marty Schmude: Basic Programming with the Casio ClassPad, First Edition, http://www.casio.edu.shriro.com.au/products/classpad/pdf/programming_cp.pdf

[7] Immo O. Kerner, Christian Wagenknecht: Einführung in das Strukturierte Programmieren, BASIC und Kleincomputernutzung; Manuskriptdruck Berlin 1989

[8] Immo O. Kerner: Informatik Klasse 11, Berlin 1990

[9] http://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm

[10] Immo O. Kerner: Numerische Mathematik mit Kleinstrechnern. Mathematik für Lehrer Band 18. Berlin 1985. **ISBN-13:** 978-3326003979

[11] Bautsch: Strukturierte Programmierung http://de.wikibooks.org/wiki/Strukturierte_Programmierung

[12] H.-O. Peitgen u.a.: Bausteine des Chaos Fraktale, Springer-Verlag Berlin Heidelberg

New York 1992, **ISBN** 3-540-55781-4, S. 353ff.

[13] E. Specht, R. Strich: geometria - scientiae atlantis. Otto-von-Guericke-Universität Magdeburg 2009, **ISBN** 978-3-940961-19-8

[14] Alfred Schilling, Wolfgang Töpfer: Informatik. Lehrbuch für das strukturierte Programmieren, Berlin 1988, **ISBN-13:** 978-3060625024

[15] Gilbert Greefrath, Jens Weitendorf: „Modellieren mit digitalen Werkzeugen“ in Borromeo Ferri, R., Greefrath, G. und Kaiser, G Hrsg.: Mathematisches Modellieren für Schule und Hochschule, Springer Spektrum 2013, **ISBN**: 978-3-658-01579-4, S. 189 ff

[16] F. Mosteller: Fifty Challenging Problems in Probability with Solutions, New York 1965, **ISBN**: 0-486-65355-2, Seite 73ff.

[17] Ruppert, M., 2013: „Eingesperrte Zufallspunkte“ in Ruppert, M. und Wörler, J. Hrsg.: Technologien im Mathematikunterricht, Springer Spektrum 2013, **ISBN**: 978-3-658-03007-0