

Der Krieg der Kerne - Grundlagen zur Spieleprogrammierung

# Von Knirpsen und Gnomen

von Martin Rogge

**Kampf bis aufs Messer - bis der Gegner um Gnade winselt. Mit diesem Kurs wollen wir das Prinzip der »Kampfprogramme« verdeutlichen.**

**D**ie Zelle ist auf dem besten Weg, den Speicher zu übernehmen. Gerade hat sie einen »Nihilisten« überschrieben. Dem Gegner bleiben nur noch wenige Züge. Halt, was ist das? Ein »Knirps« greift an! Wird sich das Blatt jetzt wenden?

So könnte eine typische Szene im »Krieg der Kerne« aussehen. Dieser »Krieg« findet jedoch nicht auf dem Schlachtfeld statt: Selbst geschaffene und erprobte »Kämpfer«, kleine Programme, fechten im C64 Scharmützel aus. Ganz nebenbei lernt jeder Spieler die Grundlagen der Assembler-Programmierung kennen.

»Redcode«, so nennt man die einfache Programmiersprache, welche die Grundlage für solche »Kernkriege« bildet. Dies ist keine Programmiersprache für allgemeine Anwendungen, sondern speziell zur Entwicklung von Kampfprogrammen gedacht.

## Speicher ohne Ende

Sie, Ihre Freunde und Bekannten können die besten Programme gegenseitig in die Computerarena schicken. Diese wird durch einen simulierten »Ringspeicher« mit 1000 Speicherzellen repräsentiert. »Ringspeicher« deshalb, weil Anfang und Ende des Speicherbereichs miteinander verkettet sind. Beide Kampfprogramme stehen in dieser Speicherarena.

Die Basis fürs Kampfspiel bildet der leistungsfähige Interpreter MARS (Memory Array Recode Simulator). MARS verwaltet den Speicherbereich und arbeitet die in ihm enthaltenen Programme quasi parallel ab. Nur elf Befehle benötigt er dafür, daher der Name für die Programmiersprache: (Redcode=REDuced CODE, verringerter Befehlssatz).

## So wird gekämpft

Jedes Programm hat die Aufgabe, den Gegner auszulöschen. Dazu kann es sich hemmungslos selbst kopieren, vermehren, Granaten werfen, Feuerwalzen durch den Speicher schicken und vieles mehr. Andererseits kann man sich gegen solche Nettigkeiten auch hervorragend schützen. Da während des Kampfes der Speicherinhalt auf dem Bildschirm ab-

gebildet wird, kann man seinen Programmen dabei zuschauen. Übrigens ist das Prinzip vom Krieg der Kerne schon lange Zeit bekannt (s. »Spektrum der Wissenschaft«, Ausgabe 8/84). Aktuell geworden ist der Kernkrieg durch das gleichnamige Public-Domain-Programm für den Atari ST. Allerdings vermißt man darin die entscheidende Fähigkeit des Multitasking.

Das Programmpaket besteht aus drei Teilen:  
- KRIEG DER KERNE (Ladeprogramm in Basic),  
- KERN.BAS (Hauptprogramm),  
- KERN.OBJ (Maschinenspracheroutinen).

Laden Sie den »Kleinkrieg« von der beiliegenden Diskette:  
LOAD "KRIEG DER KERNE",8

Gestartet wird durch die Eingabe von RUN. Die beiden anderen Programme werden nachgeladen.

Für Interessierte hier die Speicherbelegung (nach Aktivierung des Programms):

Die Grenze des Basic-Speichers wird auf \$6800 herabgesetzt. Von \$6801 bis \$6FFF liegt »KERN.OBJ«, von \$7000 bis \$9FFF der Datenbereich.

## Das Hauptmenü

Nach dem Start meldet sich das Basic-Programm mit dem Hauptmenü (Abb. 1). Die gewünschten Funktionen lassen sich durch Druck auf die entsprechende Zahlentaste aktivieren:

### KRIEG DER KERNE

- 1 - EDITOR
- 2 - KÄMPFER LADEN
- 3 - KÄMPFER SPEICHERN
- 4 - KAMPF STARTEN
- 5 - SPEICHER LISTEN
- 6 - ENDE

[1] Das Hauptmenü nach dem Laden des Programms

### 1 - Editor

Hier wird der Quelltext der Kampfprogramme bearbeitet. Es handelt sich um einen einfachen Zeileneditor, der ASCII-Dateien verarbeitet (Tabelle). Sie können auch Ihr gewohntes



Textverarbeitungsprogramm dafür verwenden und die Datei später als Kampfprogramm laden.

## 2 - Kämpfer laden

Eine ASCII-Datei wird nach Angabe des Namens in den entsprechenden Textspeicher (Speicher A oder B) geladen. Auf der Diskette zum Sonderheft finden Sie folgende Dateien: *Nihilist*, *Knirps*, *Gnom*, *Roggen*, *Zelle*, *Chang 1*, *Sprengbrand*.

## 3 - Kämpfer speichern

Der Inhalt des jeweiligen Textspeichers liegt als sequentielle Datei (Kennung SEQ) vor. Bei der Namenseingabe im ASCII-File wird der Zeile ein Anführungszeichen vorangestellt, damit sich auch Sonderzeichen wie Kommata verwenden lassen.

## 4 - Kampf starten

Die Inhalte der beiden Textspeicher werden in einen Zwischencode übersetzt (compiliert), die Anzahl der Züge bis zum »Unentschieden« sowie die der Kämpfe abgefragt. Ist kein Fehler aufgetreten, beginnt der Kampf nach der letzten Eingabe. Dabei sehen Sie den Ringspeicher auf dem Bildschirm. »Blau« entspricht dabei Programm A, »Rot« Programm B. Sind alle Kämpfe durchlaufen, bekommt als Signal der Rahmen eine graue Farbe. Nach Druck auf die RETURN-Taste sehen Sie das Ergebnis der Computerschlacht, ein zweites <RETURN> führt ins Hauptmenü zurück.

## 5 - Speicher listen

Der Inhalt des Ringspeichers wird erneut in Redcode übersetzt und auf dem Bildschirm ausgegeben. Blau dargestellte Speicherzellen entsprechen wieder Kämpfer A, rote Kämpfer dem Typ B. Durch Betätigen der Taste »Pfeil links« können Sie diesen Menüpunkt abbrechen.

## 6 - Ende

Damit verlassen Sie das Programm, ohne die ursprüngliche Speicherverteilung wiederherzustellen und den von »KERN.OBJ« benutzten Speicherbereich freizugeben. Ein Neustart ist also sofort möglich. Um in den Einschaltzustand zu kommen, genügt ein Reset oder die Anweisung »SYS 64738« (Warmstart).

## Kampfprogramme richtig eingesetzt

Bevor wir den Redcode näher beschreiben wollen, noch ein paar Zeilen darüber, wie die Kampfprogramme eingesetzt werden. Wir zeigen Ihnen den Weg vom Editor (Abb. 2) bis zum »Kriegsausbruch«.

Der Inhalt beider Textspeicher ist als einfache ASCII-Datei abgelegt, die beim Starten in einen Zwischencode übersetzt (compiliert) wird. Diesen Code überträgt das Hauptprogramm – sofern kein Fehler auftritt – in den für Maschinenprogramme reservierten Bereich und führt ihn aus. Mögliche Fehler sind beispielsweise ein leerer Textspeicher, ein falscher Befehl, eine ungültige Adressierungsart usw. Es ist übrigens erlaubt, im Quelltext Kommentare einzufügen. Diese müssen, durch ein Leerzeichen getrennt, hinter einem Befehl stehen.

War die Übersetzung (Compilierung) fehlerfrei, so muß der Computer die Anzahl der Züge pro Kampf wissen. Der vorgeschlagene Wert für die maximale Zugzahl (10000) ist für Testzwecke gedacht. Im Wertungskampf dürfen es natürlich mehr sein (z.B. 30000). Manche Programme retten oft ein Unentschieden über die Zeit, indem sie sich perfekt »einigeln«. Bei einer höheren Zugzahl sind auch häufig Barrieren zu durchbrechen. Maximal sind 65535 Züge erlaubt, mehr sind programmtechnisch nicht durchführbar – der C64 ist schließlich nur ein 8-Bit-Computer.

Anschließend fragt Sie das Programm nach der Anzahl der Kämpfe. Um ein objektives Urteil zu erhalten und Zufallssiege auszuschließen, sollte man mindestens zehn, besser noch 100 Durchgänge spielen. Erst dann sieht man, ob ein Kampfprogramm gegen einen guten Spieler bestehen kann.

<E>	führt zurück ins Hauptmenü
<+> und <->	Vor- und Rückwärtsblättern des Quellcodes um jeweils 10 Zeilen.
<U>	Wurde das Kampfprogramm geladen, erhält es den Dateinamen als Quelltext-Name zugeordnet. Mit <U> kann man ihn verändern oder neu schreiben.
<D>	Der Quelltext wird auf Drucker ausgegeben, die Geräteadresse ist 4, Sekundäradresse ist 0.
<S>	Schreiben: Auf die Frage nach der Zeilennummer gibt man die Zeile an, ab der der neue Code stehen soll. Eventuell nachfolgende Zeilen werden automatisch nach hinten verschoben (entspricht einem Insert-Modus). Man kann keine höhere Zeilennummer eingeben als die dem letzten Programmschritt folgende. Verlassen wird der Programmier-Modus durch die Eingabe einer leeren Zeile.
<L>	Löschen von Zeilen. Bei der Eingabe wird die erste und letzte zu löschende Zeile verlangt. Will man bis zum Programm löschen, gibt man als Endzeile eine entsprechend hohe Zahl ein, beispielsweise 999.
<A>	Ändern einer Zeile. Natürlich kann immer nur eine der zehn sichtbaren Zeilen geändert werden. Vorher muß also mit <+> oder <-> geblättert werden.

Tabelle. Alle Editorkommandos mit Erläuterungen

Zur Initialisierung der Arena wird Kämpfer A an der absoluten Adresse 0 in den Ringspeicher gebracht und bekommt die Farbe »Blau« als Kennung. Kämpfer B (rot) besetzt eine zufällig ausgewählte andere Position, ohne »A« zu überschreiben. Dann starten beide Kampfprogramme und arbeiten abwechselnd je einen Befehl ab. Programm A erhält also den relativ geringen Vorteil des ersten Zugrechts. Ein Kampfprogramm ist zerstört, wenn MARS keinen gültigen Code mehr vorfindet. Ansonsten wird die Partie nach der zuvor angegebenen Zugzahl mit »Unentschieden« beendet.

Ist die angegebene Zahl der Kämpfe ausgeführt, färbt sich der Bildschirmrahmen grau und der Computer wartet auf die RETURN-Taste, nach dessen Druck er das Ergebnis anzeigt.

## Die Befehle der Kampfsprache

Kommen wir zu den Befehlen, der Programmierung in Redcode:

### DAT z

DAT ist kein ausführbarer Befehl, sondern legt einfach die Zahl z als Datenwert ab. Sie darf Werte von -999 bis +999 annehmen. Trifft MARS auf einen Datenwert, gilt das aktuelle Task (s. SPL) als geschlagen.

### MOV w a

Der Move-Befehl kopiert den Wert w an die Adresse a. Dabei sind folgende Adressierungsarten zulässig:

– *Immediate*, kenntlich gemacht durch das vorangestellte Rautenzeichen <#>. »#3« bedeutet, daß der Wert 3 an die Zieladresse kopiert wird.

– *Direct*: Hier geben Sie eine Adresse an, z.B. »3«. Dies bedeutet, daß die Speicherstelle, die drei Adressen hinter dem



MOV-Befehl steht, den zu transportierenden Wert enthält. Findet das Programm an entsprechender Position einen Befehl, ist dessen Kopie wiederum der ausführbare Code. Da der Ringspeicher zyklisch aufgebaut ist, gibt es keine absoluten Adressen.

- *Indirect*, kenntlich gemacht durch einen Klammeraffen <@>. »@-5« bedeutet, daß zunächst aus der Speicherstelle fünf Adressen vor dem MOV-Befehl ein Datenwert geholt wird (kein Befehl). Dieser Datenwert stellt die Adresse dar (wieder relativ zum MOV-Befehl), in welcher der eigentliche Datenwert bzw. Befehl steht, der zur Zieladresse transportiert werden soll.

**NAME : ZELLE**

0	JMP	14
1	DAT	0
2	MOV	0-1 011
3	CMP	-2 #0
4	JMP	4
5	SUB	#1 -4
6	SUB	#1 7
7	JMP	-5
8	ADD	#15 5
9	MOV	4 09

  

**SCHREIBEN  
LOESCHEN  
ÄNDERN  
DRUCKEN  
UMBENENNEN  
+/-  
ENDE**

[2] Der Editor  
zum  
»Basteln«  
und Verändern  
eigener  
Kampfprogramme

Für die Zieladresse a sind nur die Adressierungsarten »Direct« und »Indirect« zulässig. So bedeutet z.B. »MOV @-2, @-1«, daß der Inhalt aus der Speicherposition des Datenwerts von Adresse -2 an die Speicherposition des Datenwerts von Adresse -1 kopiert wird.

Die Adressierungsarten für w und a entsprechen bei allen Befehlen dem gleichen Schema.

## ADD w a

Der Wert w wird zur Adresse a addiert.

## SUB w a

Ein Wert w wird von der Adresse a subtrahiert.

## JMP a

Springe zur Adresse a.

## JMZ a w

Springe zur Adresse a, wenn w = 0.

## JMN a w

Springe zur Adresse a, wenn w < > 0.

## DJZ a1 a2

Erniedrige Adresse a2 um »1« und springe zur Adresse a1, wenn a2 = 0.

## DJN a1 a2

wie DJZ, Sprung erfolgt bei a2 < > 0.

## CMP w w

Vergleiche beide Werte. Sind sie ungleich, so überspringe die nächste Anweisung.

## SPL a

»Split« ist der interessanteste Befehl. SPL legt einen neuen »Task« an. Tasks sind eigenständig lauffähige Programme, die

nebeneinander abgearbeitet werden. Mit dem Befehl SPL entsteht quasi ein Unterprogramm, das neben der alten Befehlslinie abgearbeitet wird. Der alte Task durchläuft das Programm an der nächsten Adresse weiter. Der neue Task beginnt ab Adresse a mit der Abarbeitung. Bis zu 1000 Tasks können gleichzeitig aktiv sein, dann werden keine mehr zugelassen. Jeder dieser Tasks läuft dann natürlich nur ein Tausendstel so schnell wie ein einziger Task, da alle nacheinander zyklisch abgearbeitet werden. Gelöscht wird ein Task, wenn er auf eine DAT-Anweisung trifft.

Das ist schon der gesamte Befehlssatz von Redcode. Sämtliche Befehle belegen nur eine Speicherstelle zusammen mit allen Operanden, der Zahlenbereich von 0 bis 999 ist erlaubt.

## So entwickelt man Kampfprogramme

Jetzt zum praktischen Teil, der Programmierung unserer Computerkämpfer. Das einfachste Programm, der »Nihilist«, sieht so aus:

```
JMP 0
```

Der Nihilist tut nichts anderes, als ständig den JMP-Befehl auszuführen. Mit dem Parameter »0« ist nicht die absolute Adresse 0 des Ringspeichers gemeint, sondern die entsprechende Speicherstelle relativ zum JMP-Befehl: 0 Speicherstellen davon entfernt steht eben dieser Befehl selbst. Übrigens weiß kein Redcode-Programm genau, an welcher absoluten Beginnadresse es sich im Ringspeicher befindet, da alle Adressen stets relativ zum aktiven Befehl gelten. Der »Nihilist« ist damit eine hervorragende Defensivwaffe.

Wesentlich aggressiver stellt sich der »Knirps« dar:

```
MOV 0 1
```

Dieser »Mini-Kämpfer« ist die schnellste Angriffswaffe überhaupt. Er kopiert sich immer an die nächste Adresse, die ihn dann beim nächsten Schritt enthält. Es läuft also ein Band von »Mov 0 1« mit maximaler Geschwindigkeit durch den Speicher. Um ihn zu stoppen, könnte man folgendes programmieren:

```
MOV #0 -1  
JMP -1
```

Es wird immer eine »0« in die Speicherstelle vor den MOV-Befehl geschrieben, in der Hoffnung, daß der Knirps dort hineinläuft und überschrieben wird. Das Stopper-Programm benötigt zwei Operationen, um die Schleife einmal zu durchlaufen. Ein Single-Task-Knirps schafft in dieser Zeit zwei Schritte und kann so den Stopper locker überschreiben. Ein Knirpswerfer muß sich unbedingt gegen eigene Knirpse schützen. Wenn das Programm jeden Knirps mit einem Task startet, muß die Knirpsfalle doppelt so schnell sein: Zwei Tasks müssen diese Knirpsfalle abarbeiten. Gegen feindliche Knirpse, die schneller als die eigenen sind (weil weniger gegnerische Task laufen), ist diese »Falle« machtlos.

```
SPL 3  
MOV #0 -1  
JMP -1  
SPL -2  
etc.
```

Bevor wir in unserem Kampfprogrammkurs weitermachen, sollten Sie noch den »Gnom« kennenlernen, ein Beispiel für



die indirekte Adressierung. Der Gnom ist ein übler Granatwerfer:

```
ADD #5 3
MOV #0@2
JMP -2
DAT -2
```

Der ADD-Befehl erhöht das Argument der DAT-Anweisung um »5«, dort entsteht eine »3«. Der MOV-Befehl schreibt eine »0« in die Adresse 3, welche die DAT-Zeile angibt. Dies ist die Speicherzelle nach dem DAT. Beim nächsten Schleifendurchlauf wird die Speicherzelle 8 beschrieben usw. Wichtig ist, daß der Zähler nach Erreichen des Wertes »998« mit »+5« wieder zu »3« wird, damit der Gnom keine Granaten auf sich selbst wirft. Darauf muß der Programmierer achten.

Kehren wir zur Knirpsfalle zurück: Sie brauchen diese im Programm »Roggen«, das so heißt, weil sich Knirpse wie die »Getreidegrannen« über den Bildschirm ziehen. Zuerst ist eine Knirpsfalle enthalten, dann der Granatwerfer, der Knirpse verschießt und startet:

```
SPL 3
MOV #0 -1
JMP -1
SPL -2
MOV 6@5
SUB #2 4
SPL @3
ADD #271 2
JMP -4
DAT 550
MOV 0 1
```

Da der SPL-Befehl zwei Zeilen unter dem MOV-Befehl steht, muß vom Zähler (DAT-Zeile) zuerst »2« subtrahiert werden, bevor der SPL-Befehl diesen nutzen kann: Die Adresse ist ja relativ zum aktiven Befehl um »2« kleiner geworden.

[3] Beispiel  
für einen  
in Redcode  
zurückverwandten  
Speicherinhalt

```
47 MOV 0 1
48 SUB #1 7
49 JMP -5
50 ADD #15 5
51 MOV 4 C9
52 SPL C8
53 ADD #653 2
54 JNZ 2 -11
55 DAT 150
56 MOV -1 4
57 SUB #11 3
58 MOV #15 -15
59 JMP -15
60 MOV 0 1
61 MOV 0 1
62 MOV 0 1
63 MOV 0 1
64 MOV 0 1
65 MOV 0 1
66 MOV 0 1
67 MOV 0 1
68 MOV 0 1
```

<CR> DRUECKEN

Der Lebensweg des Programms (ohne Feindeinwirkung) sieht so aus: Der Speicher wird mit Knirpsen gefüllt, die alle in die Knirpsfalle am Anfang des Roggens geraten. Schließlich schlägt ein Knirps genau im MOV-Befehl des Roggens ein, er wird selbst zum Knirps. Die noch aktive Knirpsfalle frißt alle verbliebenen Knirpse auf.

Ein ganz anderes Programm ist die »Zelle«: Sie teilt sich. Aus einer werden zwei, dann vier, acht, 16 usw. Irgendwann überschreiben sich die Zellen jedoch selbst. Obwohl die Zelle eine Selbstmordbedingung enthält, bilden sich »Krebszellen«, erkennbar auf dem Bildschirm an einer Punkt-Lücke-

Punkt-Reihe. Werden allerdings vom Gegner (beispielsweise einem Roggen) viele Zellen getötet, kann es vorkommen, daß überhaupt keine Krebszellen auftreten. Auf jeden Fall ergeben sich interessante Kämpfe. Abbildung 3 zeigt den Inhalt des Ringspeichers nach einem Zweikampf zwischen dem Roggen und der Zelle.

```
JMP 14
DAT 0
MOV @-1 @11
CMP -2 #0
JMP 4
SUB #1 -4
SUB #1 7
JMP -5
ADD #15 5
MOV 4 @9
SPL @8
ADD #653 2
JMP 2 -11
DAT 848
MOB -1 4
SUB #11 3
MOV #15 -15
JMP -15
```

Im ersten Teil wird die Zelle als Ganzes kopiert, dann der Zähler (zu Beginn »848«) berichtigt und einzeln kopiert. Anschließend wird die Kopie als neuer Task gestartet, der Zähler um »653« erhöht und die Zelle neu gestartet – allerdings nur dann, wenn der Schleifenzähler noch den Wert »0« besitzt. Weshalb er geändert sein könnte? Ganz einfach: Vielleicht hat der Gegner eine andere Zelle dort hineinkopiert oder Knirpse ausgeschickt oder, oder...

## Strategien

Soweit unsere Einführung in Redcode. Man kann übrigens ganze Strategien in einen Kämpfer stecken, beispielsweise erst Granaten werfen, dann Knirpse aussenden und sich anschließend in die Defensive begeben.

Die Strategien sind damit jedoch noch lange nicht erschöpft. Probieren Sie die vorgestellten Beispielprogramme aus. Die besonderen Tricks dieser Kämpfer lernen Sie aus der Analyse (Menüpunkt 5, Speicher listen) am besten kennen. Verwenden Sie dann »gelungene« Abschnitte in Ihren eigenen Programmabläufen. Empfehlenswert ist, Ihre besten Kämpfer mit verschiedenen Gegnern zusammenzuführen. Dadurch lernen Sie Schwächen und Stärken der kleinen Helden kennen. Treten Sie anschließend gegen die »Kunstgeschöpfe« Ihrer Freunde an, um sich im ehrlichen Wettkampf zu messen. Sie werden feststellen, wieviel Spaß es macht, die Kampfkraft Ihrer Helden ständig zu steigern.

Zum Abschluß ein Tip: Nach der Eingabe von POKE 27083,0 wird nur Kämpfer A bedient. Das Austesten geht doppelt so schnell und vor allem störungsfrei vor sich. POKE 27083,255 stellt den ursprünglichen Zustand wieder her. Viel Vergnügen in der Welt der »Computerschlächten«. (bl)

## Kurzinfo: Krieg der Kerne

**Programmart:** Lernprogramm für Spielabläufe im Computer

**Laden:** LOAD "KRIEG DER KERNE".8

**Starten:** nach dem Laden RUN eingeben

**Steuerung:** Tastatur

**Besonderheiten:** Erläuterung von Multitasking-Programmierung und Assembler-Sequenzen

**Benötigte Blocks:** 28

**Programmautor:** Martin Rogge