Spektrum.de

01.01.1993

Neues vom Krieg der Kerne

Das 1984 von A. K. Dewdney kreierte Computerspiel ,"Krieg der Kerne" hat sich weiterentwickelt. Es gibt neue Standards – und neue Erweiterungen dazu.

Helge Robitzsch

Was einmal als harmloses Spielchen anfing, ist mittlerweile zu einer bösen Landplage ausgeartet: Computerviren, jene kleinen Programme, die in übler Absicht so geschrieben sind, daß sie sich an existierende Programme anhängen, sich vermehren, indem sie Kopien ihrer selbst im Speicher der befallenen Rechner ablegen, und je nach der kriminellen Energie ihres Autors sich durch triumphierende Sprüche auf dem Bildschirm oder auch durch Zerstörung sämtlicher Datenbestände auf der Festplatte bemerkbar machen.

Mittlerweile bekämpft man die Computerviren mit ihren eigenen Waffen: Die sogenannten Viruskillerprogramme durchsuchen den Speicher des Computers und löschen alle Virusprogramme, die sie erkennen. Daraufhin setzen die Virenautoren ihre Bemühungen daran, ihre Produkte mutationsfähig zu machen, so daß sie – jedenfalls zunächst – nicht wiederzuerkennen sind.

Das ist zwar alles bitterer Ernst; aber dennoch: Programme zu schreiben, die ähnlich wie Virus- und Viruskillerprogramme in einem Computer gegeneinander kämpfen, ist eine faszinierende Vorstellung. Alexander K. Dewdney, langjähriger Autor der Rubrik ,,Computer-Kurzweil“ in dieser Zeitschrift, hat sie unter dem Namen ,,Krieg der Kerne“ (Core War) realisiert und beschrieben (Spektrum der Wissenschaft, August 1984, Seite 8, November 1984, Seite 16, Mai 1985, Seite 8, April 1987, Seite 8).

Selbstverständlich will niemand aggressive Killerprogramme frei in seinem Rechner herumvagabundieren lassen. Beim Krieg der Kerne bekämpfen sich deshalb zwei Programme in einem abgesicherten Speicherbereich, der sogenannten Arena, unter der Aufsicht des speziellen Überwachungsprogramms MARS (Memory Array Redcode Simulator). Sie sind in einer besonderen

Programmiersprache namens Redcode (reduced code) zu schreiben, die zur Klasse der einfachen Assemblersprachen gehört und in der ursprünglichen Fassung neun, später elf verschiedene Typen von Anweisungen enthielt.

Als das Spiel mit den kämpfenden Programmen immer beliebter wurde und man sogar dazu überging, Weltmeisterschaften auszutragen, wurde eine internationale Gesellschaft ins Leben gerufen (The International Core War Society), die Bemühungen um eine Vereinheitlichung der Spielregeln bündelte und schließlich 1988 die „Core War Standards '88" publizierte. Dort ist unter anderem festgelegt, aus welchen Anweisungen die Sprache Redcode besteht, was bei ihrer Ausführung in der Arena geschieht und wie eine Entscheidung über Sieg und Niederlage zu treffen ist (Kasten Seite 12).

Der Standard ist noch nicht alt, hat sich aber für den Erfindungsreichtum der Programmierer bereits als zu eng erwiesen. Erweiterungen erlauben noch erheblich raffiniertere Strategien.

Kämpfer und Killer

Zu einem Kampf werden zwei in Redcode formulierte Programme in die Arena gebracht, wo sie abwechselnd jeweils eine der von den Programmierern vorgesehenen Anweisungen ausführen. Ein Programm erfüllt zwar vom Standpunkt seines Autors aus einen gewissen Zweck; vom Standpunkt der Maschine gesehen ist es jedoch – wie jede andere Datei – nichts weiter als eine Folge von Bits. Und was einem ordentlichen Programmierer ein Greuel ist, macht hier den Reiz des Spiels aus: Jedes der Programme ist während des Ablaufs manipulierbar – durch sich selbst oder durch seinen Gegner.

Ein Kampfprogramm kann die unterschiedlichsten – defensiven oder auch aggressiven – Strategien realisieren. Ziel ist in jedem Fall, dem gegnerischen Programm den Garaus zu machen, indem man es auf eine sogenannte Bombe auflaufen läßt. Es handelt sich um eine spezielle, nicht ausführbare Anweisung, die DAT-Anweisung. Besondere Würze erhält ein Kampf dadurch, daß sich ein Kämpfer mit Hilfe der SPL-Anweisung in mehrere Prozesse aufspalten kann, wodurch er sozusagen mehrere Leben erhält. Ein Kampf ist erst dann verloren, wenn alle Prozesse eines Kämpfers tot sind. Aus Gründen der Fairness kommt nach der Ausführung einer Anweisung immer der Gegner an die Reihe, auch wenn ein Kämpfer über mehrere Prozesse gebietet.

Die Arena besteht aus hintereinanderliegenden gleichartigen Speicherzellen, die durch fortlaufende Zahlen (0, 1, 2 und so weiter) adressiert werden. Hierin unterscheidet sich die Arena nicht von einem üblichen Computerspeicher. Sie hat traditionsgemäß eine Länge von 8192 Adressen und ist ringförmig geschlossen, so daß nach der Adresse 8191 wieder die Adresse 0 folgt. In jede Speicherzelle paßt genau eine Redcode-Anweisung.

Zu den einfachsten Kampfprogrammen zählen der schon von Dewdney vorgestellte KNIRPS und die KNIRPS-FALLE (Kasten Seite 14). Erfahrene Redcode-Programmierer werden entdecken, daß die Sprache durch einige Konstrukte, die klassischen Assemblersprachen entlehnt sind, besser lesbar und weniger fehleranfällig geworden ist. Aber auch einem Neuling dürfte es nicht schwerfallen, nachzuvollziehen, was sich beim Kampf der beiden Zwerge in der Arena im einzelnen abspielt. Nur soviel sei verraten: KNIRPS wandert Schritt für Schritt im Speicher voran, eine Spur von KNIRPSen hinter sich lassend. KNIRPS-FALLE wartet in aller Seelenruhe auf einen angreifenden KNIRPS, wobei sie ständig die Speicherzelle in ihrem Rücken mit DAT-Anweisungen bombardiert. Wer gewinnt? KNIRPS gewinnt nie, da er dem Gegner keine DAT-Anweisung in den Weg legen kann. KNIRPS-FALLE gewinnt nur dann, wenn entweder der Abstand der Kämpfer im Speicher – genauer: die Differenz der Adressen ihrer Einsprungpunkte – gerade ist und KNIRPS beginnt oder wenn umgekehrt der Abstand ungerade ist und KNIRPS-FALLE beginnt. In allen anderen Fällen verwandelt KNIRPS die KNIRPS-FALLE in einen weiteren KNIRPS, und beide laufen bis in alle Ewigkeit einträchtig durch den Speicher. Damit man in endlicher Zeit zu einem Ergebnis kommt, gilt ein Kampf als unentschieden, wenn jeder Kämpfer 30000 Anweisungen ausgeführt hat und immer noch kein Sieger feststeht.

Ein weiterer relativ primitiver Kämpfer ist GNOM: Er überdeckt die ganze Arena mit einem Teppich aus Bomben, dessen Lücken so berechnet sind, daß er selbst verschont bleibt. Raffiniertere Programme können festellen, ob sich ein Angreifer naht, und ihm entfliehen, indem sie sich selbst an eine andere Stelle der Arena kopieren.

Überleben durch Prozesse

Als besonders erfolgreich hat sich die Strategie erwiesen, Kopien des eigenen Programms an immer

neuen Positionen in der Arena abzulegen und sie mit Hilfe der SPL-Anweisung als eigenständige Prozesse zu aktivieren. Das Programm MÄUSE (MICE) gewann mit dieser hemmungslosen Vermehrung die erste Core-War-Weltmeisterschaft (Bilder 1 und 2 links).

Der wesentliche Teil von MÄUSE ist die Kopierschleife, die von dem Label Kopier bis zur DJN-Anweisung reicht. Damit reproduziert sich MÄUSE selbst in einem anderen Bereich der Arena. Um den Kopiervorgang unablässig zu wiederholen, schließt eine weitere Schleife von Start bis zur JMZ-Anweisung die Kopierschleife ein. Darin wird jeweils nach dem Kopieren die Kopie durch die SPL-Anweisung als neuer Prozeß gestartet und mit der ADD-Anweisung eine neue Adresse als Kopierziel eingestellt. Die Verwendung von JMZ anstelle von JMP stellt sicher, daß eine Maus sich nur dann weitervermehrt, wenn die Laufvariable Zaehler nicht durch Feindeinwirkung oder andere MÄUSE verändert ist. Eine angeschlagene Maus begeht Selbstmord durch Ausführen der nachfolgenden DAT-Anweisung. Dadurch steigt für den Kämpfer mit der MÄUSEpopulation die Chance, den Kampf mit gesunden MÄUSEn fortzusetzen.

Es ist nicht einfach, Kämpfer zu entwickeln, die dieses Programm regelmäßig besiegen können. Versuchen Sie es! Entwerfen Sie ein Redcode-Programm KATZE, das am liebsten MÄUSE frißt! Wenn es Ihnen gelingt, dann schicken Sie es über den Verlag an den Autor; Spektrum der Wissenschaft wird darüber berichten.

Die MARS-Maschine

Es ist sinnvoll, sich als Schauplatz des Kriegs der Kerne einen eigenen, recht primitiven Computer namens MARS vorzustellen. Dieser ist aus rein praktischen Gründen (wer will sich schon eigens für dieses Spiel einen Computer zusammenlöten?) in den umgebenden Rechner eingebettet, indem ein Programm innerhalb des Wirtsrechners die Wirkung seiner Komponenten modelliert. Ein Blick in die Struktur dieses virtuellen Computers ist schon deswegen interessant, weil er – mit einer Variante – das Prinzip des klassischen, 1946 von Arthur W. Burks, Herman H. Goldstine und John von Neumann vorgeschlagenen Rechners in Reinform verwirklicht. Die meisten heutigen – nicht-parallelen – Computer arbeiten nach diesem Prinzip, wenngleich der Benutzer sich dank einer Fülle von Anwendungsprogrammen in der Regel darüber keine Gedanken mehr machen muß. Allerdings kann ein

von-Neumann-Rechner in der Urform nur jeweils ein Programm abarbeiten. MARS ist insofern etwas moderner, als er zwei Programme zu steuern vermag, die ihrerseits mit der SPL-Anweisung noch Tochterprozesse starten können.

Der MARS-Rechner (Bild 2 rechts) besteht aus einem Speicher (der Arena), in dem sich die Programme mit ihren Daten befinden, zwei Warteschlangen, in denen die Prozesse der beiden Kämpfer verwaltet werden, einem Rechenwerk, das die arithmetischen und logischen Operationen einzelner Redcode-Anweisungen ausführt, und schließlich einer Kontrolleinheit, welche die jeweils näch-ste auszuführende Anweisung bereitstellt, eventuell vorverarbeitet (abhängig von der Adressierungsart) und an das Rechenwerk zur Ausführung übergibt.

Eine Warteschlange (queue) ist ein Datenspeicher aus einer Folge gleichartiger Komponenten. Wie eine echte Warteschlange am Fahrkartenschalter ändert sie sich nur dadurch, daß an einem Ende (der Eingabeseite) weitere Komponenten angefügt oder am anderen Ende (der Ausgabeseite) Komponenten weggenommen werden. Was zuerst angefügt worden ist, kommt auch zuerst wieder heraus. Man bezeichnet Warteschlangen daher auch als first-in-first-out-Speicher, abgekürzt FIFO. Die Warteschlangen von MARS bestehen aus den Adressen von Anweisungen, die zur Ausführung vorgemerkt werden.

MARS hat auch eine Verbindung zur Außenwelt, um die kämpfenden Programme entgegenzunehmen und den Verlauf des Kampfes zurückzumelden. An dieser Schnittstelle sitzt zweckmäßig ein weiteres Programm, das Überwachungsprogramm ZEUS. Auch wenn diese im Standard gewählte Namensgebung nicht ganz konsequent ist (Mars gehört zur römischen, Zeus zur griechischen Götterwelt), soll sie hier beibehalten werden. ZEUS erteilt nicht nur Befehle an MARS, sondern hilft dem Anwender auch bei der Formulierung von Kampfprogrammen, der graphischen Darstellung der Ergebnisse und der Fehlersuche. Insbesondere bei der Entwicklung neuer Kämpfer kann es nützlich sein, den Ablauf der Programme Schritt für Schritt am Bildschirm zu verfolgen. Schnittstelle und Überwachungsprogramm sind im Core-War-Standard nicht festgelegt.

Das Zusammenwirken der Komponenten von MARS ist einer näheren Betrachtung wert. Normalerweise bringt MARS die Kämpfer zu Beginn an zufällige Positionen in der Arena, wobei ein vorgegebener

Minimalabstand einzuhalten ist. Damit es nicht zu lange dauert, bis der Kampf spannend wird, setzen wir diesmal zwei kleine Kämpfer, einen defensiven und einen aggressiven, dicht aneinander in die Arena (Bild 3). Kämpfer 1 belegt die Adressen 2000 und 2001, Kämpfer 2 die Adressen 2002 bis 2004. Zur besseren Lesbarkeit ist der Inhalt der Speicherplätze in Redcode angegeben, obwohl die Anweisungen in irgendeiner Weise codiert im Speicher liegen können (der Redcode-Standard legt die Codierung nicht fest). Jede der beiden Warteschlangen enthält zu Anfang die Speicheradresse, mit der die Ausführung des zugehörigen Kämpfers begonnen werden soll (Zustand 0). Von der Arena ist nur der interessante Ausschnitt zwischen den Adressen 1999 und 2004 wiedergegeben. Nicht von den Kämpfern belegte Speicherzellen erhalten zu Beginn den Inhalt DAT #0 #0. (Im Gegensatz zu der hier verwendeten Core-War-Version schreibt der Standard die Initialisierung des leeren Speichers mit DAT 0 0 vor. Da diese Anweisung wegen der direkten Adressierung illegal ist, kann ein Kampfprogramm eine Speicherzelle dieses Inhalts als unbeschrieben erkennen.)

Durch Zufallsauswahl wird festgelegt, welcher Kämpfer beginnt. Wir nehmen an, dies sei Kämpfer 2. Dann entfernt die Kontrolleinheit aus der Warteschlange 2 die Adresse 2003 und kopiert sich die an dieser Adresse im Speicher liegende Redcode-Anweisung MOV @−1 < −1. In der Kopie ersetzt sie zunächst irgendwelche in der Anweisung vorhandene indirekte Adressierungen durch die effektiven Adressen. Die so bereinigte Form der Anweisung lautet nun MOV (2002) (2001), wobei die Klammern die effektiven Adressen kennzeichnen. Als Nebeneffekt der Adressierungsart prädekrement-indirekt (<) wird das B-Feld der DAT-Anweisung an Adresse 2002 um eins erniedrigt. Die bereinigte MOV-Anweisung wird an das Rechenwerk zur Ausführung übergeben. Die Ausführung einer Anweisung verändert stets die Arena oder die betreffende Warteschlange oder beides. Im Beispiel wird zunächst die Adresse der nächsten Speicherzelle 2004 an die Warteschlange 2 angehängt, dann die an der Adresse 2002 liegende DAT-Anweisung in die Speicherzelle 2001 kopiert (Zustand 1).

Nach Ausführung der ersten Anweisung von Kämpfer 2 ist Kämpfer 1 an der Reihe. Abgesehen davon, daß für ihn die Warteschlange 1 zuständig ist, ist der Ablauf völlig analog dem für Kämpfer 2 beschriebenen. So wird immerfort abwechselnd für beide Kämpfer die Adresse der auszuführenden Anweisung aus der entsprechenden Warteschlange weggenommen und die Adresse der nächsten Anweisung hinten an die Warteschlange angehängt. Was ist die nächste Anweisung? Im Normalfall nach dem von-Neumann-Prinzip die mit der nächsthöheren Adresse. Wenn jedoch bei einer

Sprunganweisung die Sprungbedingung erfüllt ist, kommt die Adresse des Sprungziels in die Warteschlange.

Eine weitere Ausnahme bilden die Anweisungen SPL und DAT. Die Anweisung SPL hat ja den Sinn, einen weiteren Prozeß zu eröffnen. Also wird zuerst die in der SPL-Anweisung angegebene Adresse und sofort danach die Adresse der auf die SPL-Anweisung folgenden Speicherzelle an die Warteschlange angehängt. Da vorher die Adresse der Anweisung selbst aus der Warteschlange entfernt wurde, hat sich die Länge der Warteschlange insgesamt um eins erhöht: Ein neuer Prozeß ist geboren. Im Gegensatz dazu bewirkt die Ausführung der DAT-Anweisung nur das Entfernen der eigenen Adresse, aber keinerlei neuen Eintrag in die Warteschlange: Der Prozeß hat ein Leben verloren. Stößt die Kontrolleinheit auf eine leere Warteschlange, so hat der zugehörige Kämpfer sein letztes Leben ausgehaucht und damit den Kampf verloren.

Lassen Sie mit Bleistift und Papier die MARS-Maschine mit den beiden Kämpfern weiterlaufen! Der Kampf ist nach der Ausführung von genau 12 Anweisungen durch das Rechenwerk zu Ende (Zustand 13). Gewonnen hat – und das ist nicht verwunderlich – der aggressivere Kämpfer 2.

Spannung und Faszination des Kriegs der Kerne kommen erst dann so richtig zur Wirkung, wenn man den Kampfverlauf durch graphische Darstellung der Arena am Bildschirm hautnah miterleben kann. Am besten stellt man die Arena als rechteckiges Gitter von Punkten oder kleinen Quadraten dar. Jeder Gitterpunkt repräsentiert eine Speicherzelle, links oben beginnend mit Adresse 0 bis zur höchsten Adresse rechts unten. Anhand der Färbung einer Zelle ist erkennbar, welcher Kämpfer in die Zelle zuletzt geschrieben hat. Man beobachtet Geländegewinne, Ausschwärmen, Wandern von KNIRPSen, Vermehrung von MÄUSEn und andere beeindruckende Vorgänge (Bilder 1 und 4).

Weiterentwicklungen

In neuen Varianten des Spiels kann man andere Anweisungen einführen oder die MARS-Maschine um Register erweitern, die einem Kämpfer Informationen über sich und den Gegner liefern. So hat beispielsweise ein Programm die Möglichkeit festzustellen, ob es selbst oder sein Gegner durch eine Bombe getroffen wurde (Bild 4), und seine Strategie davon abhängig zu machen. Programme mit

intelligenten Angriffsstrategien können den Gegner gezielt an seinen Schwachpunkten attackieren.

Es gibt auch bereits Varianten von MARS, die zu Beginn mehr als zwei Kämpfer in die Arena laden. Reihum wird je eine Anweisung ausgeführt, wobei für jeden Kämpfer eine eigene Warteschlange zur Verwaltung seiner Prozesse vorhanden ist. Was von diesen Versuchen Eingang in den Core-War-Standard findet, bleibt abzuwarten.

Auch die modernen Konzepte der Rechnervernetzung und der Parallelverarbeitung könnten das Spiel bereichern. Der Kampf findet etwa in dem Modell eines Rechnernetzes oder einer hierarchischen Baumstruktur von Rechnerkomponenten statt, wobei Anweisungen und Prozesse nicht unbedingt nacheinander, sondern teilweise parallel ausgeführt werden. Ziel des Spiels ist, alle Rechnerkomponenten zu beherrschen.

Ein völlig anderes Spiel kann man entwerfen, wenn man sich von dem Konzept des von-Neumann-Rechners löst und den Kampf beispielsweise in einem künstlichen neuronalen Netz toben läßt (vergleiche „Wie neuronale Netze aus Erfahrung lernen“, Spektrum der Wissenschaft, November 1992, Seite 134). Eines Tages werden vielleicht die Spieler versuchen, die „Gedanken“ und „Träume“ eines künstlichen Gehirns zu beeinflussen.

Aus: Spektrum der Wissenschaft 1 / 1993, Seite 10

Helge Robitzsch