Computer machen keine Fehler

Chrilly Donninger auf den Spuren eines Gerüchts

Schach ist ein Hasardspiel, meinte einst der für seine spitze Feder bekannte holländische Großmeister Jan Hein Donner. Die Nimzo-Werkstatt nahm sich die Worte des verehrten Meisters zu Herzen und wechselte beim Wiener Open 1998 ins Glücksspielgeschäft. Jeder konnte für zehn Schilling Einsatz gegen Nimzo blitzen. Das Geld kam in einen Topf. Ein etwaiger Nimzotöter konnte mit dem Obolus seiner weniger glücklichen Vorgänger nach Hause ziehen.

Für Großmeister Igor Glek rollte die Kugel nach Wunsch. Zusätzlich zum Scheck für den ersten Platz beim Open knackte er den mit 700 Schilling schon prall gefüllten Nimzo-Jackpot. Er war sichtlich zufrieden, daß ihm dies ausgerechnet mit der Glek-Variante in der Wiener Partie gelungen war. Die 69 Vorgänger hatten auf die falsche Zahl gesetzt. Die Partie wurde mit dem Standardkommentar: »Wahnsinn, so ein Fehler, dabei bin ich schon (fast) auf Gewinn gestanden. Ja, ja, Computer machen halt keine Fehler, Menschen schon« beendet. Nimzo hat sich wahrscheinlich über den ersten Teil der Aussage amüsiert, ich über den zweiten.

Programmierer sind nicht der liebe Gott. Und selbst bei dem könnte man sich fragen, ob er nicht am letzten Tag, im Projekt-Abschluß-Streß, ein paar Programmfehler eingebaut hat. Schachprogrammierer haben meist auch keine Eva, der sie das Schlamassel in die Schuhe schieben können. Auch die These, sie haben dem Programm eine Entscheidungsfreiheit eingebaut, wirkt nicht sehr überzeugend. Sie können sich daher nur Asche aufs Haupt streuen und sich mit der Einsicht »Menschen machen halt Fehler« trösten.

Die Eva in der Programmiergeschichte war ein kleiner Käfer. Als die erste Programmiererin, Admiral Grace Hopper, das erste Computerprogramm fertiggestellt hatte, wollte es partout nicht das tun, was es tun sollte. Nach langem verzweifelten Suchen fand Mrs. (Admiral wurde sie erst in reiferen Jahren) Hopper die Ursache. Ein zerquetschter Käfer zwischen zwei Relais des Rechners hatte die Programmlogik durcheinandergebracht. Kritische Geister bezweifeln den Wahrheitsgehalt der Geschichte. Aber erstens war Admiral Hopper eine reizende Dame, die hinreißend Geschichten erzählen konnte, und zweitens ist der Mythos immer interessanter als die banale Wahrheit. Programmierfehler werden bis zum heutigen Tag mit liebevollem Abscheu »Bugs« (engl. Käfer, Wanze, Laus) genannt. Programme zur Beseitigung von Programmierfehlern heißen dementsprechend auch Debugger (»Entlauser«).

Nimzo98 war mein viertes Schachprogramm. Nachdem der Mensch angeblich aus seinen Fehlern lernt, beschloß ich, die bei der Entwicklung auftretenden Fehler zu notieren. Insgesamt habe ich 182 Fehler niedergeschrieben. (Falls sie glücklicher Nimzo98-Besitzer sind, keine Panik, diese 182 Fehler wurden *vor* der Auslieferung des Programms entdeckt).

Viele Wege führen nach Rom

Prinzipiell gibt es zwei verschiedene Möglichkeiten, dem Computer Anweisungen zu geben. Man kann dies entweder in einer sogenannten Hochsprache oder in der Maschinensprache tun. Hochsprachen sind mehr dem menschlichen Denken (genaugenommen dem Denken von Programmierern) angepaßt. Durch einen maschinellen Übersetzer (engl. Compiler) werden die Befehle dann in die Maschinensprache übertragen. Ein Befehl der Hochsprache bewirkt in der Regel mehrere Rechenschritte im Prozessor. Zumindest theoretisch kann ein in einer Hochsprache geschriebenes Programm in die Maschinensprache verschiedenster Rechner übertragen werden. Ein PC-Programm kann dadurch etwa auch auf einem Mac zum Laufen gebracht werden. Die heute gebräuchlichste Hochsprache ist »C« bzw. die Weiterentwicklung »C++« (gesprochen »C-Plus-Plus«).

Die alternative und traditionelle Methode besteht in der direkten Programmierung in Maschinensprache, vulgo Assembler. Darin beschreibt man Schritt für Schritt die Operationen des Prozessors bis ins kleinste Detail. Um so zu programmieren, muß man sich quasi in das Innenleben des jeweiligen Prozessors versetzen. Ein Assembler-Programm ist in der Regel fünfmal so lang wie das entsprechende in C. Es ist daher wesentlich einfacher und schneller, ein Programm in einer Hochsprache zu schreiben. Laut Informatik-Schulbuch-Weisheit verbricht man in einer Hochsprache auch wesentlich weniger Fehler als in Assembler. Das wird daher heute fast nicht mehr verwendet – so besteht z.B. der Code von Windows95 zu mehr als 99,9% aus C bzw. C++-Programmteilen, und nur ein verschwindender Rest ist in Assembler geschrieben.

Schachprogrammierung ist noch eine der letzten Assembler-Bastionen. Die alten Assembler-Hasen Frans Morsch, Ed Schröder, Richard Lang und meine Wenigkeit zeigen der C-Jugend noch immer, wie man mit der Geiß ackert. Assemblerprogrammierung ist zwar mühsam und langwierig, man kann damit aber auch gleichzeitig die letzten Reserven aus dem Prozessor herausholen. In Assembler geschriebene Programme sind daher schneller als das C-Pendant.

Eine wesentliche Frage in meiner Nimzo98-Fehlerstudie war daher: Ist ein Assembler-Programm wirklich eine Bug-Brutstätte? Von den 182 gefundenen Käfern waren ganze 26 sogenannte »dumme Assemblerfehler«. Die sind mit Tippfehlern in einem Artikel vergleichbar. Diese Fehler würde man in einem C-Programm nicht machen, weil einem bereits der Übersetzer/Compiler auf die Finger klopft. Weitere 18 Assembler-Fehler kamen durch besonders aggressive Optimierung des Programms zustande. Zum Beispiel wurde die Position des Königs von einer vorhergehenden Berechnung weiterverwendet und nicht neu vom Hauptspeicher in den Prozessor eingelesen. Dadurch erspart man sich pro Position ein paar Nanosekunden. Wenn nun der entsprechende Programmteil auch durch eine andere Befehlssequenz erreicht werden kann, steht der König am falschen Feld. Solche Fehler passieren auch in einem C-Programm. Auf Grund der begrenzten Möglichkeiten ist es in C aber wesentlich schwieriger, auf diese Art und Weise Schaden anzurichten.

Operationen, die pro Position nur einmal durchgeführt werden, sind in einem Schachprogramm nicht besonders zeitkritisch. Daher sind in Nimzo98 die Programmteile zur Verwaltung des Eröffnungsbuches und das sogenannte Orakel in C geschrieben. Interessanterweise enthielten diese Teile ebenfalls 13 »dumme« und acht »Optimierungs«-Fehler. Compiler sind wie ein Rechtschreib-Korrektur-Programm: Sie erkennen zwar einfache Tippfehler, es rutschen ihnen aber z.B. Beistrich- oder Grammatikfehler durch. Das Verhältnis 26 zu 13 bzw. 18 zu 8 entsprach ungefähr der Aufgabenverteilung zwischen Assembler- und C-Programmtext. Für einen alten Hasen stimmt die Schulbuchweisheit »Assembler pfui, Hochsprache hui« nicht.

Der Weiß-Schwarz-Bug

Eine besondere Spezies von »dummer Fehler« zieht sich durch die Nimzo-Geschichte. Dies ist der von mir so getaufte Weiß-Schwarz-Bug. Für ein bestimmtes schachliches Konzept (z.B. Freibauern, Vorposten) schreibe ich immer zuerst die Programmteile für Weiß. Danach wird der Programmtext kopiert, und im nächsten Schritt werden die Farben und Vorzeichen vertauscht. Der Weiß-Schwarz-Bug besteht nun darin, daß bei dieser Farben- und Vorzeichenumkehr etwas übersehen bzw. nicht richtig vertauscht wird. Nachdem dieser Käfer ein guter alter Freund des Hauses ist, habe ich bei Nimzo98 besonders aufgepaßt, und er trat daher nur fünfmal auf. Er wird uns aber in den Partiebeispielen im zweiten Teil des Artikels noch begegnen.

Die bisherigen 70 Fehler haben eines gemeinsam: Man schreibt oder sagt dem Computer etwas anderes als man eigentlich sagen will. Die restlichen 112 Fehler sind von gänzlich anderer Natur: Der Computer befolgt genau alle ausgedachten Regeln. Diese sind aber falsch bzw. unvollständig. Meist treten diese Denkfehler beim unerwarteten Zusammenspiel von Spezialfällen auf.

An »En passant« kommt keiner vorbei

Die klassische Quelle für Denkfehler ist En passant. Die Programmteile zur Behandlung von En passant machen weniger als 1% des Gesamtprogramms aus. Trotzdem haben sich in diesen Teil 31 Denkfehler eingeschlichen! Es ist wirklich unglaublich, was man bei der En-passant-Behandlung alles falsch machen kann.

So hatte ich zunächst folgende Regeln programmiert:

Regel 1: »Wenn die Seite am Zug nicht im Schach steht, braucht man für die Legalität eines Zuges nur zu prüfen, ob die ziehende Figur nicht gefesselt ist«.

Regel 2: »Wenn man im Schach steht, gibt es prinzipiell folgende Zugmöglichkeiten: a) Man zieht auf das Feld der zuletzt gezogenen Figur und schlägt dadurch die schachgebende Figur des Gegners. b) Man stellt – bei langschrittigen Angreifern – eine Figur dazwischen. c) Der König zieht auf ein unbedrohtes Feld.«

Bevor Sie weiterlesen, sollten Sie sich überlegen, welchen En-passant-Fehler die Regeln jeweils enthalten (wenn Sie schnell darauf kommen, sollten Sie die nächste Nimzo-Version an meiner Statt programmieren).

Nun, Regel 1 beachtet nicht, daß bei einem En-passant-Zug auch der gegnerische En-passant-Bauer »gefesselt« sein kann. Man muß bei einem En-passant-Zug noch zusätzlich prüfen, ob der eigene König nach dem Schlagen des En-passant-Bauern nicht durch eine gegnerische Figur bedroht wird.

Regel 2 übersieht ebenfalls ein kleines, aber feines En-passant-Detail. Falls die schachgebende Figur ein Bauer im Doppelschritt war, kann man den Angreifer nicht nur durch Ziehen auf das Feld des Angreifers, sondern auch durch En passant schlagen.

Regel 1 hat bei mir gleich zweimal zugeschlagen. Wenn man festgestellt hat, daß En passant wegen der »Fesselung« des gegnerischen En passant-Bauern

nicht möglich ist, muß man nicht nur den Zug des eigenen Bauern zurücknehmen, sondern auch den En passant-Bauern wieder auf das Brett stellen.

Möglicherweise werden Sie nun einwenden: »Ich spiele nun schon ewig Schach, aber eine Stellung mit gefesseltem En passant-Bauer ist mir noch nie untergekommen«. Wahrscheinlich haben Sie bisher ein paar tausend Stellungen am Brett gehabt. Nimzo berechnet/untersucht aber während einer einzigen Turnierpartie ein paar Milliarden Stellungen. Einem Schachprogramm laufen daher »bizarre« Stellungen wesentlich häufiger über den Weg als einem menschlichen Spieler. Macht sich ein illegaler Zug unter eine Milliarde legaler Züge bemerkbar? Wenn man Glück hat, schon. Nimzo nimmt z.B. immer an, daß der vorhergehende Zug legal war. Es wird daher beim aktuellen Zug nicht mehr überprüft, ob eine geschlagene Figur der gegnerische König ist. Durch Vermeidung dieser Abfrage werden ein paar Nanosekunden pro Stellung eingespart. Im Falle des »gefesselten En-passant-Bauern« verschwindet daher im nächsten Zug der König vom Brett. Dies bewirkt weitere Fehler, und das Programm verabschiedet sich.

Der Fehler kann aber auch jahrelang gut getarnt in seinem Kokon sitzen. Er entfaltet sich erst unter günstigen Bedingungen, z.B. bei einem besonders wichtigen Turnier, zu seiner vollen Pracht.

Ich habe vor einiger Zeit das Kapitel »Wir schreiben ein Schachprogramm« für das Buch »Schach am PC« geschrieben. In diesem Beitrag wurde das inzwischen unter Programmierern berühmte Chrillysche En-passant-Fehler-Gesetz formuliert: »Wenn Sie beim Testen Ihres Programms keinen En-passant-Fehler gefunden haben, wird dieser sich beim ersten wichtigen Turnier bemerkbar machen«. ChessBase-Chef Matthias Wüllenweber hat es noch um die folgende Schlußfolgerung erweitert: »Man muß ein Programm mindestens so lange testen, bis man einen En passant-Fehler gefunden hat«.

Weitere Spezialfälle beim Schach sind die Umwandlung und die Rochade. Sie sind zwar nicht ganz so fehlerträchtig wie En passant, aber mit 20 bzw. 13 Fehlern waren sie auch ein bevorzugtes Käfer-Brutgebiet.

So wurde für die Rochade folgende Regel verwendet: »Ein König auf e1 besitzt noch sein Rochaderecht, wenn er von e1 nicht weggezogen ist«. Beginnt ein Spiel aus der Grundstellung, dann ist diese Regel auch hieb- und stichfest. Der Benutzer kann aber eine Stellung eingeben, in der der König auf e2 steht. Fährt nun in Folge der König von e2 nach e1, dann ist er von e1 noch *nicht* weggezogen, das Rochaderecht ist trotzdem verwirkt.

Die restlichen Fehler verteilen sich relativ gleichmäßig auf den übrigen Programmtext. In den Spezialfällen En passant, Rochade und Umwandlung habe ich im Mittel pro vier Zeilen Programmtext einen Fehler gefunden. In den übrigen Teilen wurde nur alle 300 Zeilen ein logischer Käfer aufgespürt (dazu kommen allerdings noch die 70 »dummen« Fehler).

Lernt man aus seinen Fehlern?

Nimzo98 wurde im Sommer 1997 geschrieben. Im Mai 1998 habe ich für ein elektronisches Reiseschach eine Nimzo-Light-Version erstellt. Für dieses Programm hatte ich nur drei KByte Speicher zur Verfügung, außerdem mußte ich es für einen gänzlich anderen Prozessor umschreiben. Das Projekt war eine Feuerwehr-Aktion. Der ursprünglich dafür vorgesehene Programmierer hatte es nicht geschafft, in drei KByte ein funktionsfähiges und vor allem stark spielendes Programm unterzubringen. Als ich mit der Programmierung begann, sollte das Programm schon fertig sein. Es interessierte mich natürlich, ob ich von den Nimzo98-Fehlern gelernt hatte und ob z.B. weniger En-passant-Fehler auftauchen würden. Die Antwort ist ein klares Nein. Es wurde wie eh und je hauptsächlich En passant-, Umwandlungs- und Rochade-Ungeziefer aufgespürt. Das gesamte Produktionsteam hatte allerdings im Sinne der Wüllenweberschen Folgerung etwas gelernt: Diese Programmteile wurden besonders intensiv geprüft. Es ist daher nicht gänzlich unmöglich, daß im Endprodukt kein En-passant-Fehler mehr gefunden wird.

Benutzer-Fehler

NimzoWerkstatt-Teilhaber Helmut Weigel arbeitete lange Zeit für den einst renommierten Tischcomputer-Hersteller Hegener&Glaser. Eines Tages rief ein erboster Kunde an und beschwerte sich bitter, daß das sündteure Mephisto-Spitzenprogramm einen illegalen Bauernzug ausführe. Wie sich herausstellte, war das Programm ausnahmsweise unschuldig. Der gute Mann spielte nach eigenen Angaben seit Jahren Turnierschach, die En passant-Regel war an ihm aber unbemerkt vorbeigegangen.

Kokons

Die bisher beschriebenen Fehler wurden alle bereits bei der Programmerstellung bzw. in der anschließenden Testphase erkannt und beseitigt. Sie kommen dem Käufer eines Programms nicht mehr in die Quere. Manche Käfer puppen sich aber ein und schlüpfen manchmal erst nach Jahren. Im Folgenden möchte ich ein paar schöne Schmetterlinge präsentieren.

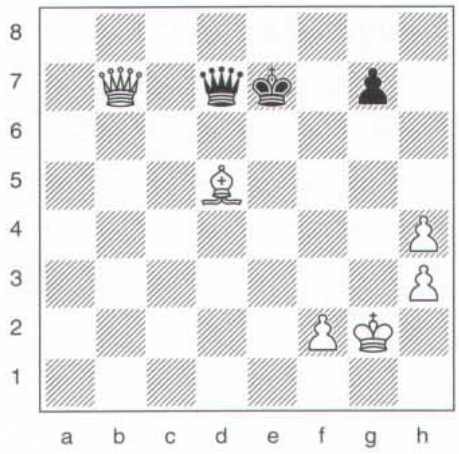
It's not a bug, it's a standard

Unter Bergsteigern kursieren so markige Sprüche wie »wer hinunterfällt, ist nur zu blöd zum Anhalten«. Nach überstandener Todesgefahr sprechen Bergsteiger vom »Probeliegen im Holzpyjama«. Durch Verniedlichung der Gefahr wird die Angst besänftigt. Programmierer wenden denselben Trick an. Bereits der Ausdruck »bug« ist eine Verniedlichung. Eine weitere beliebte Redensart ist »it's not a bug, it's a feature« (Es ist kein Fehler, sondern eine Programmeigenschaft«). Die Firma ChessBase hat dieses Prinzip in ihrer Datenbank noch perfektioniert. Das sogenannte CBF-Format war jahrelang der de facto-Standard bei der Aufzeichnung von Turnierpartien (inzwischen ist CBF von ChessBase selbst durch das CBH-Format ersetzt worden). Im CBF Format werden – um Platz zu sparen – Züge nicht direkt abgespeichert. Die Datenbank ruft einen einfachen Zuggenerator auf und speichert anschließend die Zugnummer. In der Ausgangsstellung wird z.B. statt »e2e4« die Zahl »5« abgespeichert, weil der Zuggenerator e2e4 als fünften Zug erzeugt. Dieser Zuggenerator hat, Sie sollten es nach dem bisher Gesagten bereits erraten, einen En-passant-Fehler. Er spielt Zylinderschach und kann von a5 nach h6 en passant schlagen. Solange die eingegebene Partie keinen derartigen illegalen Zug aufweist, wirkt sich der Fehler nicht negativ aus. Beim Dekodieren der Datenbank macht der Zuggenerator denselben Fehler, die abgespeicherte Nummer wird in den richtigen Zug rückübersetzt. ChessBase hatte damit sogar unbeabsichtigt einen guten Kopierschutz eingebaut. Jene Hacker, die das CBF-Format geknackt hatten, haben sehr lange gerätselt, warum ab und zu die Entschlüsselung von CBF-Partien fehlschlug. Des Rätsels Lösung: Ihr Zuggenerator hatte keinen En-passant-Zylinder-Bug. In Stellungen, in denen der En passant-Zug a5-h6 möglich war, stimmte die Zugnummer der ChessBase-Klones nicht mit jener des Originals überein. Jeder, der das CBF-Format lesen wollte, mußte daher in sein Programm den Zylinder-En-passant-Bug mit einbauen.

Richard Langs Vorzeichen-Schmetterling

Bei der allgemeinen Computerschach-WM 1992 in Madrid trat ich mit meinem damals noch sehr unausgereiften Programm Nimzo-Guernica an. In der ersten Runde fand das übliche Schweizer-System-Schlachten statt. Nimzo Guernica war als Opferlamm für das vielfache Mikro-Weltmeisterprogramm von Richard Lang vorgesehen. Bis zur nachfolgenden Stellung lief tatsächlich alles nach Drehbuch. Die Mitglieder des Nimzo-Teams drängten mich, die Partie doch endlich aufzugeben und statt dessen Madrid zu besichtigen. Lang-Manager Ossi Weiner hatte bereits seinen Chef Hegener telefonisch verständigt: »Erste Partie papierformgemäß gewonnen«.

Chess Genius – Nimzo-Guernica
Computer WM Madrid 1992



Weiß am Zug

Ich lehnte das Ansinnen meiner Kollegen mit dem Hinweis ab, daß man nur Briefe aufgibt. Bereits vorher hatte ich in der Genius-Hauptvariante einige Ungereimtheiten bemerkt und witterte noch eine Chance. Außerdem war der sonst so ruhige Richard Lang bereits bei den letzten Zügen ungewöhnlich nervös geworden. **56.Db4+?** »What's this?« murmelte Richard Lang kopfschüttelnd . **56...Dd6 57.De4+?** Es wird auch im zweiten Anlauf nichts mit dem Damentausch. **57...Kd8 58.Df5 Dc5 59.Dg5+ De7 60.f4.** Wieder nichts! **60...Kd7 61.Lf3 Ke8 62.Lc6+ Kf7 63.Ld5+ Kf8 64.Df5+ Ke8 65.Dc8+ Dd8 66.Lf7+ Ke7 67.De6+ Kf8 68.f5??** Es hätte noch immer ziemlich alles andere gewonnen. **68...Dd2+ 69.Kg3 Dd3+ 70.Kg4 Dd1+ 71.Kf4** (eine nette Pointe ergibt sich nach 71. Kg5 Dd2+ 72. Kh5 Dd1+ 73. Kg6 Dg4+! 74. hxg4 und Schwarz ist patt) **71... Dc1+ 72.Ke4**... Der König kommt nicht mehr aus dem Schach, die Partie endete mit Remis im 98. Zug.

Programme bekommen einen Abtausch-Bonus, wenn sie in einer materiell überlegenen Position Figuren tauschen. Dieser Bonus ist für einen Damentausch am höchsten. Richard Lang hatte irrtümlich das Vorzeichen vertauscht. Genius bekam für den Damentausch keinen Bonus, sondern einen kräftigen Malus, und vermied daher so gut es ging den Abtausch. Madrid 92 war der erste Auftritt eines Lang-PC-Programms. Ossi Weiner wurde deshalb nicht müde zu betonen, daß der Madrid-Genius noch eine Beta-Version sei und so etwas in einem käuflichen Lang-Programm nie vorkommen würde. Wie findige Leser der Zeitschrift Modul aber herausfanden, ließ sich der Fehler bis in die Lang-Tischcomputer des Jahres 1985 zurückverfolgen. Nimmt man Ossi Weiner wörtlich, dann hatten Hegener & Glaser sieben Jahre

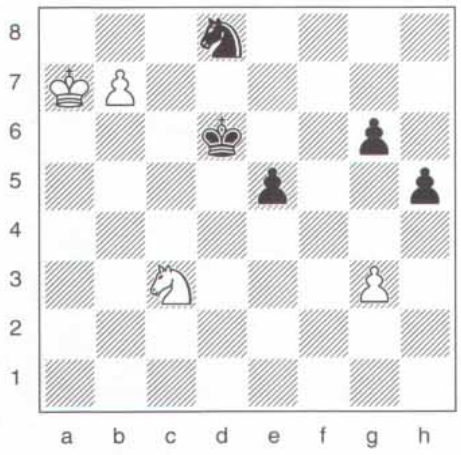
lang eine Beta-Version für teures Geld verkauft. Noch erstaunlicher ist, daß dieser elementare Fehler in sieben Jahren niemandem aufgefallen war. Der Fehler hinderte Richard Lang auch nicht daran, von 1985 bis 1992 ein paar Mal Computerschach-Weltmeister zu werden.

It's not a bug, it's a genius

Schachspieler verlieren sehr selten, weil sie einfach schlecht bzw. der Gegner besser gespielt hat. Sie verlieren nur, weil sie in (Fast)-Gewinnstellung einen Fehler gemacht haben (siehe Standardkommentar am Beginn des Artikels). Auch Computer-Programme verpassen laut Schöpfer den längst verdienten Turniersieg nur durch irgendeinen blöden Bug.

Bei der Mikro-WM 1993 in München war Nimzo-Guernica im Vergleich zu Madrid schon ordentlich gewachsen. In der zweiten Runde kam es zum Aufeinandertreffen mit dem Madrid-Champion Gideon/Rebel. Nach wechselvollem Spiel opferte Nimzo seinen letzten Springer und wickelte in die folgende Stellung ab:

Gideon – Nimzo, München 1993



Schwarz am Zug

Es folgte: **61...Sxb7 62.Kxb7 Kc5 63.Se2 Kd5 64.Sg1 e4 65.Kb6 e3 66.Kb5 Kd4 67.Kc6 Kd3 68.Sf3 e2 0-1.**

Gideon-Schöpfer Ed Schröder wurde nach der WM vom holländischen »Computerschaak« nach dem für ihn wichtigsten Ereignis von München gefragt: »Das war das Endspiel von Nimzo gegen Gideon. Weiß beurteilte die Stellung besser für sich, Schwarz sah bereits seinen Sieg. Das war unglaublich klug für ein Programm.«

Die Bewunderung von Ed war insofern berechtigt, als damalige PC-Programme keine Chance hatten, die Konsequenzen des Opfers zu errechnen. Nimzo opferte aus positionellen Überlegungen. Das Programm kannte folgende Endspiel-Regel: Wenn Weiß einen Freibauern hat, der schwarze König aus dem Quadrat ist und Schwarz keine Figur und auch keinen Freibauern besitzt, dann steht Weiß gewonnen. Weiß bekommt in dieser Stellung einen starken Gewinn-Bonus. Beim Übertragen für die schwarze Seite baute ich einen »Weiß-Schwarz-Bug« ein. Die analoge Regel für Schwarz lautete: Wenn Schwarz einen Freibauern hat, der weiße König nicht im Quadrat ist, Weiß keinen Freibauern hat und wenn *Schwarz* außerdem keine Figur besitzt, dann ist die Stellung für Schwarz gewonnen«.

Der lästige eigene Springer hinderte das Programm daran, den Gewinn-Bonus zu kassieren. Er wurde daher bei erstbester Gelegenheit geopfert. Das Computerschach-Magazin »Modul« meinte damals: »Wahrscheinlich besteht das Wesen der Genialität darin, einen Fehler einzubauen, durch den das Programm besonders gut spielt«. Als ich Ed beim nächsten Treffen die Ursache für Nimzos subtiles Spiel erzählte, war er sichtlich erleichtert, daß ich auch nur mit Wasser koche.

Wenn Ihnen also das nächste Mal ein Programm einen beeindruckend-genialen Zug vorsetzt, murmeln Sie nicht resignierend »ja, ja, Computer sehen halt alles und machen keine Fehler«. Setzen Sie sich vielmehr mit dem Programmautor in Verbindung und teilen Sie ihm mit, daß sie einen besonders heimtückischen Programmfehler gefunden haben.

Rochade-Bug in Belle

1980 gewann Ken Thompson mit *Belle* die Computerschach-WM in Linz. In der Folgezeit hat er viele aufsehenerregende Erfolge mit der Spezialmaschine erzielt. Irgendwann dazwischen entdeckte er einen erschreckenden Bug im Programmcode: Belle wollte am liebsten kurz rochieren *können* (d.h. eine Stellung, in der die Maschine noch rochieren konnte, bekam einen bestimmten Bonus). An zweiter Stelle kamen Stellungen, in denen Belle lang rochieren konnte. Etwas schlechter beurteilt wurden Stellungen, in denen kurz rochiert *war;* und den kleinsten Bonus erhielten Stellungen, in denen die Maschine bereits lang rochiert hatte. Stellungen, in denen Belle nicht mehr rochieren konnte, erhielten natürlich überhaupt keinen Bonus. Das führte dazu, daß Belle, wie man nachträglich in einigen Partien ganz deutlich erkennen konnte, erst dann rochierte, wenn abzusehen war, daß der Gegner in einer bestimmten Variante das Rochaderecht der Maschine zerstören konnte. Solange diese Gefahr nicht im Suchbaum auftauchte, rochierte Belle einfach nicht – was mitunter als besonders interessante Spieleigenschaft (Aufrechterhaltung der Spannung in der Partie) interpretiert wurde.