Klaus Boehncke, Christoph Köhler, Georg Meyer-Berg, Klaus Schulten

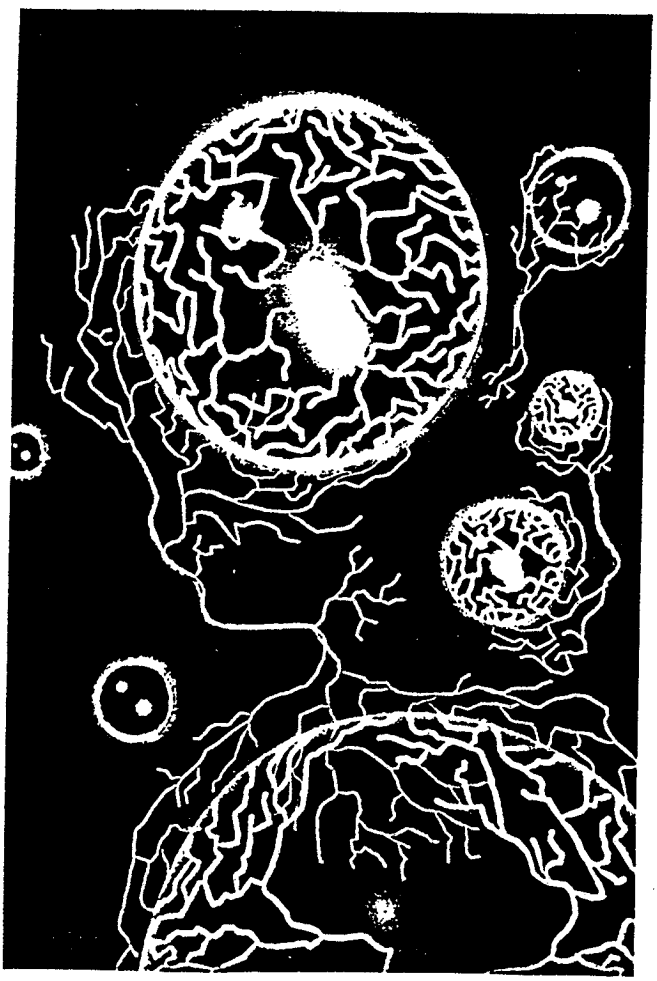
Eine Computersimulation künstlicher Wesen

Experimente in einer Welt synthetischer Psychologie

Komplexes biologisches Verhalten läßt sich durch einfache Modelle simulieren. Auf deren Grundlage (in Modula) programmierte Wesen legen ein überraschendes Verhalten an den Tag. Der Artikel schildert die Entdeckung dieser Wesen auf einem fremden Planeten in Science-Fiction-Form.

Es ist eine ungewöhnliche Umgebung, welche die Besatzung des Raumschiffes Enterprise mit Captain Kirk diesmal vorfindet. Auf einer dunklen, von äußeren Lichtquellen unbeleuchteten Ebene bewegen sich leuchtende Wesen unterschiedlicher Helligkeit in einer Weise, die für den Betrachter zunächst vollkommen unverständlich scheinen muß. Da gibt es Wesen, die in einem Gebiet vieler unbewegter Lichter von einem Lichtzentrum zum anderen gelangen, manche schlendernd, manche hastend. Andere sind scheu und scheinen Helligkeit ganz zu meiden, wieder andere aber sind neugierig. Da gibt es Wesen, die eine Lampe weiträumig einzukreisen versuchen, nahe vor ihr stehen bleiben oder nur flüchtig an ihr vorbeifahren. Es fällt eine Gruppe auf, in der die Mitglieder in ein wildes Spiel verwickelt sind; hin und wieder jedoch wird ein Mitglied vorübergehend ausgestoßen und im Dunklen stehen gelassen.

Die besondere Aufmerksamkeit der Mannschaft erregen zwei Wesen, die sich etwas abseits befinden. Ein rotes Licht versucht anscheinend mit aller Macht ein weißes zu zerstören, denn es jagt auf dieses zu, angestachelt von dem Licht in dessen Nähe besonders schnell und aggressiv. Wenn es dem weißen Licht besonders nahe kommt, versucht dieses nicht selten durch Hakenschlagen oder Flucht seinem Widersacher zu entkommen, und der aggressive Verfolger verfehlt sein Ziel. Captain Kirk befiehlt, Tracking-Kameras einzuschalten, um die Bahn der eigenartigen Lebewesen zu verfolgen. Die Bahnen von Aggressor und Feigling werden für Archivzwecke zur Enterprise gesendet (*Bild 1*). Bei einem Versuch der Mannschaft, das aggressive Wesen anzulocken und einzufangen, zerstört dieses die zu diesem Zweck eingeschaltete lichtstarke Taschenlampe bei einem zielgerichteten, ungehemmten Angriff durch Zusammenstoß.



In Dunkelheit verharrt das eingefangene Lebewesen in Bewegungslosigkeit und kann so von der Mannschaft schließlich eingefangen werden.

Die Mannschaft von Raumschiff Enterprise ist überzeugt, daß das intelligent anmutende Verhalten der Lichtwesen auf eine hohe Entwicklungsstufe schließen läßt, und beschließt, die Geschehnisse auf dem Planeten zu untersuchen. Dabei stellt eine Analyse der von der Traking-Kamera erfaßten und farblich automatisch codierten Trajektorien fest, daß die Lebewesen in ihrem Verhalten vier Grundmustern zuzuordnen sind. Entsprechende Trajektorien sind in *Bild 2* wiedergegeben. Der bereits erwähnte Aggressor ist in Bild 2 violett dargestellt. Daneben gibt es den Typ des Feiglings, der sich schon deswegen schwierig einfangen läßt, weil er sich von jeder Lichtquelle abwendet, und zwar um so schneller, je näher er dieser ist. Dieses Verhalten ist in Bild 2 als blaue Bahnkurve festgehalten. Zwei weitere Wesen erhalten bei der Mannschaft des Raumschiffes ebenfalls auf Grund ihres Verhaltens ihre Namen: der Liebhaber und der Flüchtling. Diese Typen bewegen sich im unbeleuchteten Raum mit maximaler Geschwindigkeit und erlahmen nur unter Lichteinfluß. Der Liebhaber wendet sich einem Licht bis zum Stillstand zu, während der Flüchtling durch Abkehr diesem Schicksal zu entgehen sucht. Diese Beobachtungen werden ebenfalls in Bild 2 dargestellt, und zwar mit grünen und braunen Trajektorien für Liebhaber bzw. Flüchtling. Um die Wesen möglichst ruhig zu halten, werden Aggressor und Feigling in dunklen,

Liebhaber und Flüchtling in sehr hellen Käfigen zum Raumschiff gebracht. Dr. McCoy ist ganz aufgeregt und bittet um Erlaubnis, die Wesen unters Messer nehmen zu dürfen. Schon bald hat Captain Kirk einen Bericht in den Händen:

Die Wesen sind Fahrzeuge mit einem, um eine vertikale Achse schwenkbaren, kleinen Vorderrad, zwei getrennt angetriebenen Hinterrädern, einer Lampe, zwei optischen Sensoren (Augen) und einem zwischen Sensoren und Motoren geschalteten Minigehirn ausgestattet. *Bild 10* zeigt eine Seiten- und eine Aufsicht. Jeder der beiden lichtempfindlichen Sensoren beobachtet die Lichtverhältnisse in einer anderen Raumrichtung, wobei sich die wahrgenommenen Winkelbereiche am Bug des Fahrzeuges überlappen können und meistens symmetrisch bezüglich der Fahrzeugachse liegen.

Das Ausgangssignal eines Sensors ergibt sich folgendermaßen: Der Sensor mittelt über die Intensität aller in seinem Sichtbereich befindlichen Lichtquellen und digitalisiert diesen Wert. Offensichtlich dienen die Lampen auf den Fahrzeugen dem wichtigen Zweck der Kommunikation zwischen den Lebewesen auf dem Planeten.

Schrittmotoren an den Rädern ermöglichen die Bewegung der Lebewesen. Die Motoren werden über Zuleitungen (siehe Bild 10) durch digitale Impulse vom Minigehirn angeregt und drehen dabei die Räder um einen festen Winkelbetrag vorwärts. Ansteuerung nur eines Motors bewirkt verständlicherweise eine Drehung des Fahrzeugs.

Die Umsetzung der Wahrnehmung der Umwelt in eigenes Verhalten ist Aufgabe des Gehirns. Aus seinem Aufbau ergibt sich der Charakter der Lebewesen (Aggressor, Feigling, Liebhaber, Flüchtling). Leitungen von den Sensoren und zu den Motoren, wie oben beschrieben, sind im Gehirn über sogenannte Neuroden miteinander vernetzt. Neuroden sind Schwellwertelemente, die mit beliebig vielen anderen Neuroden (eingangs- und ausgangsseitig), Sensoren (nur eingangsseitig) und Motoren (nur ausgangsseitig) fest verdrahtet sein können. Ein Neurod mit seinen Eingangs- und Ausgangsleitungen ist in *Bild 11* dargestellt.

Erreicht die Summe der gleichzeitig an einem Neurod ankommenden Impulse (Leitungen a, b, c in Bild 11) dessen Schwellwert, so feuert das Neurod, d. h. es gibt seinerseits einen Impuls ab; ansonsten liegt kein Signal an den Ausgängen des Neurods an. Ist das Neurod eingangsseitig nicht über eine exzitatorische (anregende) Leitung mit einem anderen Neurod oder Sensor verbunden, sondern über eine inhibitorische (hemmende) Verbindung (Leitung d in *Bild 11*), so trägt jedes Signal mit negativem Vorzeichen zu obiger Summe bei.

Um eine Gleichzeitigkeit von Signalen zu erreichen, arbeiten die Netzwerke taktweise, d.h. alle Neuroden und Sensoren feuern synchron. Die Verrechnung der Eingangssignale zum Wert des Ausgangspegels läuft in allen Neuroden gleich schnell ab, ihr Ergebnis steht den Ausgangskanälen erst im folgenden Zeittakt zur Verfügung. Die zur Übermittlung der Signale über die Verbindungsleitungen benötigte Zeit ist vernachlässigbar.

Grundtypen der Planetenbewohner

Sehr hoch kann die Entwicklungsstufe der Planetenbewohner anscheinend nicht sein, denn bedenkt man alle Verschaltungsmöglichkeiten, die sich mit einem System solcher Neuroden konstruieren ließen, so besitzen die entdeckten Exemplare einen doch recht einfachen Aufbau: Jeweils ein Sensor und ein Motor sind über ein einziges Neurod miteinander verbunden. Die entsprechenden Verschaltungen sind in *Bild 12* gezeigt, wobei die Neuroden durch einen Kreis und ihr Schwellwert mit der zugehörigen Zahl dargestellt sind.

Beim Aggressor und beim Feigling sind die Sensoren über Neuroden mit Schwellwert 1 an die Motoren gekoppelt. Es ist daher verständlich, warum die Fahrzeuge bei Dunkelheit, bei der von den Sensoren keine Impulse an die Neuroden weitergeleitet werden, regungslos bleiben (siehe Bild 11). Bei Lichteinfall in die Sensoren feuern die Neuroden an die Motoren und verursachen damit die Fortbewegung des Fahrzeuges entsprechend der Häufigkeit der lichtabhängigen Impulse von den Sensoren.

Beim Aggressor (siehe Bild 12) steuert der Sensor mit dem Sichtfeld zur Linken des Vehikels den rechten Motor (und umgekehrt). Solange eine einzelne Lichtquelle im Erfassungsbereich nur eines Sensors liegt, dreht sich der Aggressor dieser zu, bis beide Sensoren gleichmäßig ausgeleuchtet sind. Da jede Drehung auf Grund der Konstruktion mit einer Geradeausbewegung verbunden ist, fährt das Vehikel bei seiner Ausrichtung einen Kreisbogen. Wird die Lichtquelle von beiden Sensoren fixiert, bewegt sich das Fahrzeug auf diese zu, und zwar mit wachsender Geschwindigkeit, da die höhere Lichtintensität in der Nähe des Ziels eine höhere Impulsfolge an den Motoren hervorruft. Eine Wahrnehmung auf beiden Sensoren muß wegen der mehr oder weniger großen Überlappung des Sichtfeldes nicht bedeuten, daß die Fahrzeugachse auf das Ziel ausgerichtet ist. In diesem Fall kann ein Sensor, während der Zusteuerung auf die Lampe, diese aus dem Blickwinkel verlieren. Auf Grund der diskreten Schritte der Fortbewegung ist es außerdem wahrscheinlich, daß das Fahrzeug an der Lampe vorbeifährt. In beiden Fällen ergibt sich dann eine neuerliche Orientierung zur Lampe hin. Dies ist auch an der violett gefärbten Trajektorie in Bild 2 zu sehen.

Beim Feigling (siehe Bild 12) ist jeder Sensor mit dem Motor auf derselben Fahrzeugseite verbunden. In diesem Fall rollt das Wesen nur so lange auf das Licht zu, wie beide Sensoren dessen Quelle erkennen. Erfaßt nur noch ein Sensor das Licht, dreht sich das Vehikel weiter ab, bis eine meist vorhandene Lücke im Sichtfeld beider Sensoren das Fahrzeug jeder Bewegungsmöglichkeit beraubt, oder bis sich das Fahrzeug soweit von dem Licht entfernt hat, daß es auf Grund des Intensitätsabfalls zum Stehen kommt. Eine entsprechende Bahnkurve ist als blaue Trajektorie in Bild 2 dargestellt.

Flüchtling und Liebhaber haben die entsprechenden Verschaltungen, mit dem Unterschied, daß Schwellwert-0-Neuroden über inhibitorische Leitungen von den Sensoren angesprochen werden. Der Schwellwert ist immer erreicht, wenn keine Impulse an den Neuroden anliegen. Deshalb bewegen sich diese Typen gerade im Dunklen. Sie zeigen nur bei Absenkung der am Neurod anliegenden Impulse unter den Schwellwert 0 durch hemmende Signale seitens der Sensoren Abbremserscheinungen, die bei ungleichmäßiger Sensorausleuchtung ebenfalls zu Drehungen des Fahrzeuges vom Licht weg (Flüchtling) oder zum Licht hin (Liebhaber) führen. Auch diese Trajektorien sind in Bild 2 dargestellt, und zwar in den Farben braun bzw. grün. Ein Liebhaber muß allerdings nicht zwangsweise im Hellen verharren. Wenn es ihm durch seine optische Wahrnehmung oder durch Ungenauigkeit bei der Zusteuerung auf das angepeilte Objekt nicht mehr gelingt, beide Sensoren auszuleuchten, kann auch er (zunehmend schneller) dem Einfluß der Lampe entfliehen. Dieser Fall liegt bei der dunkelgrün gekennzeichneten Trajektorie in Bild 2 vor.

Komplexes Sozialverhalten

Nach der Lektüre von Dr. McCoy's Bericht beginnt sich Captain Kirk doch zu wundern. Woher kommt das komplexe Wechselspiel, wie es sich der Mannschaft bei Ankunft auf dem Planeten bot? Kann es durch den einfachen Aufbau der beschriebenen Grundtypen erklärt werden, wenn

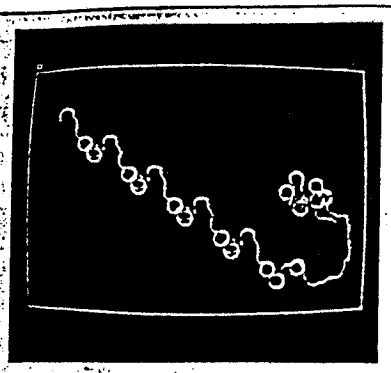


Bild 1. Verfolgung zwischen Aggressor (rot) und Feigling (weiß): Dabei ist ersterer um so aggressiver, je näher er dem Feigling kommt. Im Gegenzug dazu flieht der Feigling um so panischer, je mehr er sich bedrängt fühlt. Da der Phantasie der Wesen jedoch Grenzen gesetzt sind, ergibt sich aus dem anfänglichen chaotischen Durcheinander bald eine gewisse Ordnung.

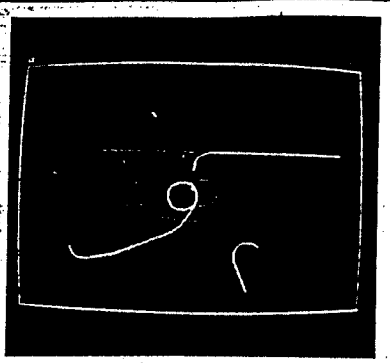


Bild 2. Typisches Verhalten der einfachsten Wesen – Feigling (blau), Aggressor (violett), Liebhaber (grün) und Flüchtling (braun): Der Feigling bewegt sich am liebsten gar nicht, nur beim Anblick einer Lichtquelle wendet er sich ab. Der Aggressor sucht mit Vorliebe helle Plätze auf und versucht die Lampen anzugreifen. Der Liebhaber steuert auf Lichtquellen zu und verharrt.

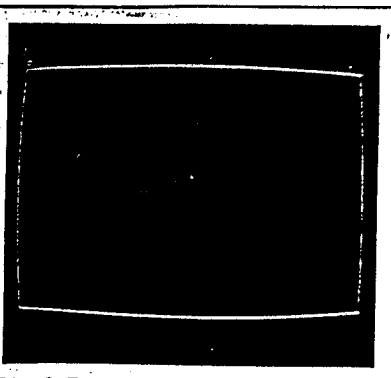


Bild 3. Das Haßliebe-Wesen – eine Synthese von Aggressor und Feigling: Dieses neugierige Wesen steuert die hellste Lichtquelle der Umgebung an und bewegt sich zögernd darauf zu. Mit abnehmender Entfernung wird es immer schneller, verbrennt sich und sucht erschrocken das Weite. Die Erfahrung hat es jedoch bald vergessen und wagt erneut einen Angriff.

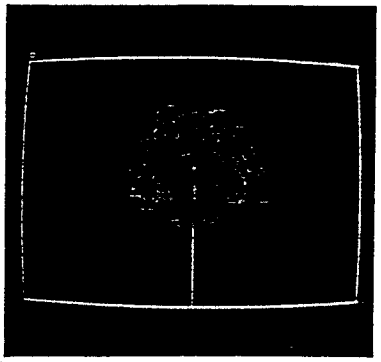


Bild 4. Das Haßliebe-Wesen nach längerer Simulationszeit: Dieses Bild zeigt ein fortentwickeltes Stadium der Anfangssituation aus Bild 3. Der „Blumenstiel" wird durch drei Liebhaber-Wesen (grün) dargestellt. Beim komplexen Haßliebe-Wesen ist trotz einer denkbar einfachen Umgebung auch nach längerer Simulation keine Periodizität erkennbar.

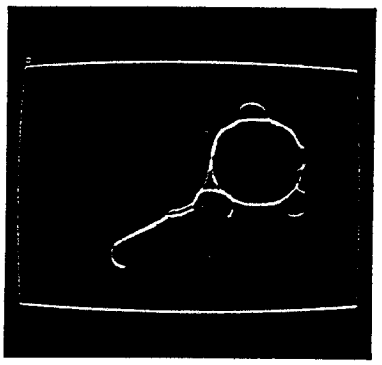


Bild 5. Auseinanderlaufende baugleiche Wesen mit ähnlichen Startbedingungen: Die weiße Bahnkurve zeigt ein Wesen, das auf einer festen Kreisbahn eine Lichtquelle trägt. Diese wird von den Aggressoren angegriffen. Interessant ist, daß geringe Abweichungen in den Startbedingungen zu großen Abweichungen im Bahnverlauf führen.

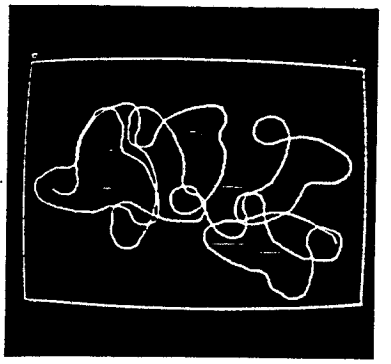


Bild 6. Zwei unterschiedliche Charaktere im Wechselspiel: Das Liebehaß-Wesen (rot) ist bestrebt, jeder Annäherung auszuweichen und wird erst aggressiv, wenn man ihm zu nahe kommt; hier verfolgt von einem Haßliebe-Fahrzeug (blau), das sich erst neugierig und zielstrebig nähert. Später jedoch erlahmt sein Interesse und es wendet sich ab.

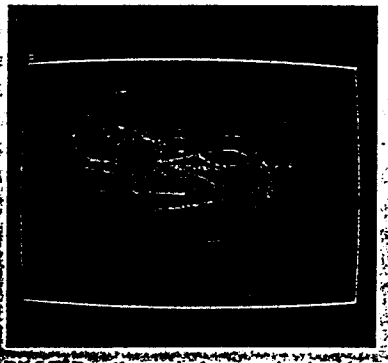


Bild 7. Pendler zwischen zwei ruhenden Lichtquellen: Dieses Wesen zeigt Ähnlichkeit zum Aggressor, da es sich meistens in der Nähe von einzelnen Lichtquellen aufhält und seltener zwischen ihnen. Anders als ein Aggressor ist es jedoch in der Lage, sich stets von einer Lichtquelle zu lösen, und zu einer weit entfernteren zu gelangen.

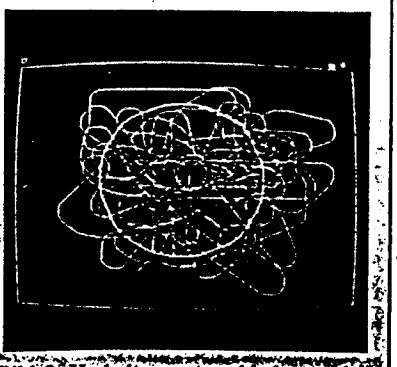


Bild 8. Verhaltensunterschiede von Aggressor und Jäger: Eine feste und eine unbeirrt auf einer Kreisbahn ziehende Lampe (beide weiß), werden von Aggressor (grün) und Jäger (violett) angegriffen. Während der Aggressor unbeirrt die stehende Lampe attackiert, sucht der Jäger nach dem „Reißen einer Beute" sogleich die nächste auf.

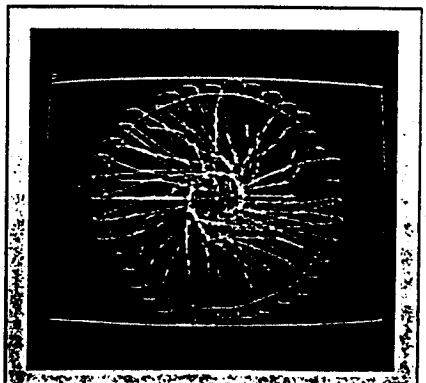


Bild 9. Pendelndes Wesen mit einer ruhenden und einer bewegten Lichtquelle: Dieses Bild zeigt einen Sammler (violett). Dieses Wesen ist in der Lage, zwischen aufgesuchter Lichtquelle und einer neuen zu unterscheiden. Anders als die bisher gezeigten Wesen entscheidet es selber, ob es sich schnell bewegen will oder nicht.

Bildschirmfotos: Benedikt Schmidt

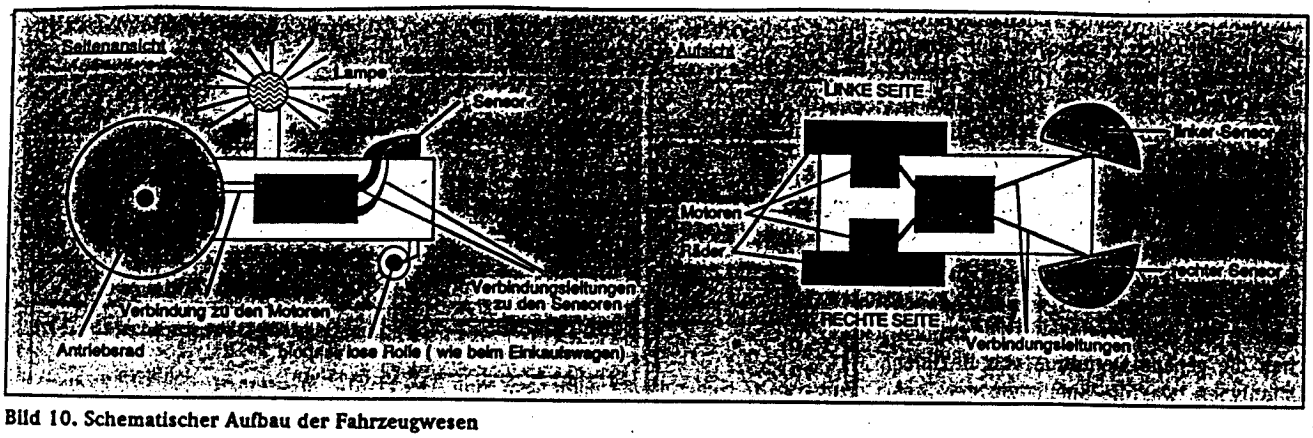


Bild 10. Schematischer Aufbau der Fahrzeugwesen

mehrere Vehikel durch ihre eigene Abstrahlung von Licht gegenseitig zur Ursache der Bewegung der anderen werden? Ein solches Beispiel hat die Tracking-Kamera im Falle der wilden Verfolgung zwischen Aggressor und Feigling in *Bild 1* erfaßt: schon bald nach Beginn der Bewegung kommt es zu einem geordneten, periodischen Wechselspiel.

Es ist einfach einzusehen, daß zufällige Wiederholungen gleicher Positionen und Ausrichtungen mit zunehmender Anzahl der miteinander wechselwirkenden Fahrzeuge unwahrscheinlicher werden oder sogar ausgeschlossen sind. Die Vielzahl der Möglichkeiten wächst aber auch wegen der Verschiedenartigkeit selbst der einfachsten Verschaltungen und der implementierbaren Sensoren (Anzahl, Ausrichtung, Überlappung, eventuell asymmetrische Anordnung, usw.) und Scheinwerfer (Anzahl, Abstrahlrichtung, Helligkeit).

Dabei sind die Möglichkeiten der Verschaltungsvielfalt auf dem vorgefundenen Planeten bei weitem noch nicht ausgeschöpft. Um der Bevölkerung der vorgefundenen Welt grausame Experimente zu ersparen, werden auf dem Raumschiff Enterprise Simulationsverfahren gesucht, mit denen nicht nur das Verhalten auf diesem Planeten modelliert werden kann, sondern auch die Lebensweise komplizierterer – vielleicht in ein paar Jahrmillionen auftretender – Lebensformen untersucht werden soll. Dem Wissenschaftsoffizier Spock kommt beim Durchstöbern der Datenbanken ein glücklicher Zufall zu Hilfe: er endeckt ein Programm, das vor vielen Jahrhunderten auf der Erde entwickelt wurde und das eine solche Welt simulieren kann. Blicken wir ihm beim Studium der Dokumentation über die Schulter.

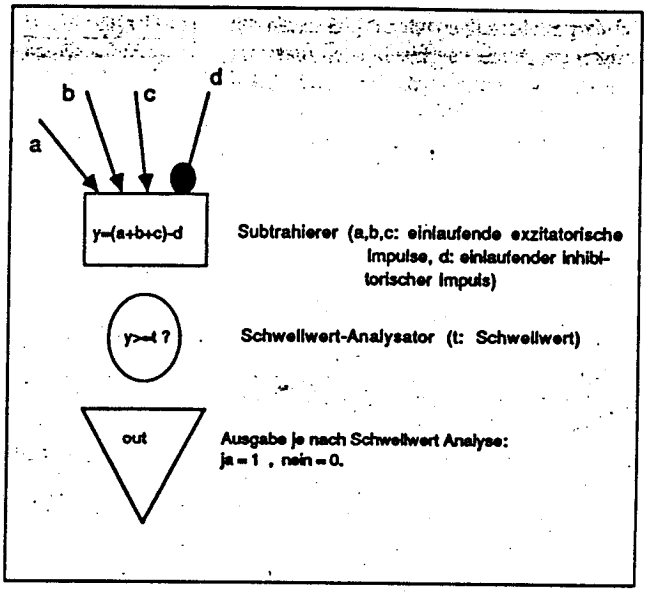


Bild 11. Funktionsweise eines Neurods

Die Braitenberg-Welt im PC

„Mitte 1987 begannen wir, angeregt durch einen Artikel im Scientific American [1],

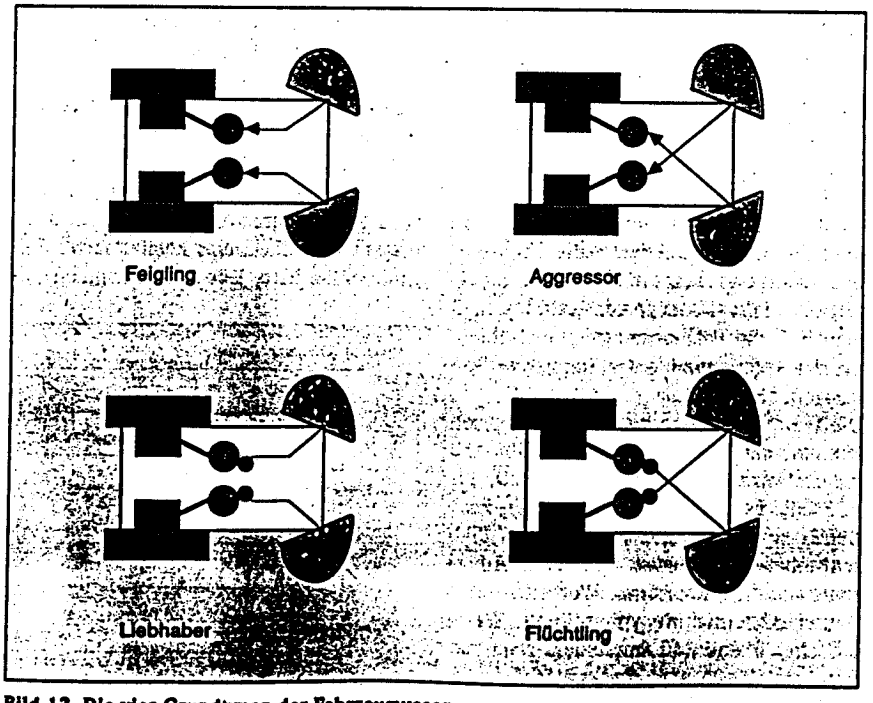


Bild 12. Die vier Grundtypen der Fahrzeugwesen

ein Seminarprojekt am Physik-Department der TU München. Es basiert auf einem Gedankenexperiment des Tübinger Professors Valentin Braitenberg [2], in dem er eine geheimnisvolle Welt erschuf. Die einfacheren Bewohner seiner Welt lassen sich ohne große Schwierigkeiten auf einem Mikrocomputer simulieren. Wichtig ist dabei, wie man die Daten über den inneren Aufbau der Wesen im Programm darstellt. Einige Programmiersprachen, darunter auch das von uns benutzte Modula-2, ermöglichen die genaue Definition von flexiblen Datentypen, was die Simulation der Braitenberg-Wesen erheblich vereinfacht.

Die Merkmale der Wesen, die wie in *Bild 10* dargestellt aufgebaut sind, müssen im Programm genau beschrieben werden, im Falle der Sensoren z. B. der durch jeden einzelnen Sensor abgedeckte Winkelbereich. Hinzu kommt für jedes Fahrzeug die aktuelle Position und die Ausrichtung der Fahrzeugachse. Alle diese Angaben müssen im Datentyp vorhanden sein, der in *Bild 13* schematisch dargestellt ist. Unser Programm „Braitenberg" ist im wesentlichen eine vereinfachte Version des Programms, das wir im Rahmen unserer Seminararbeit entwickelten.

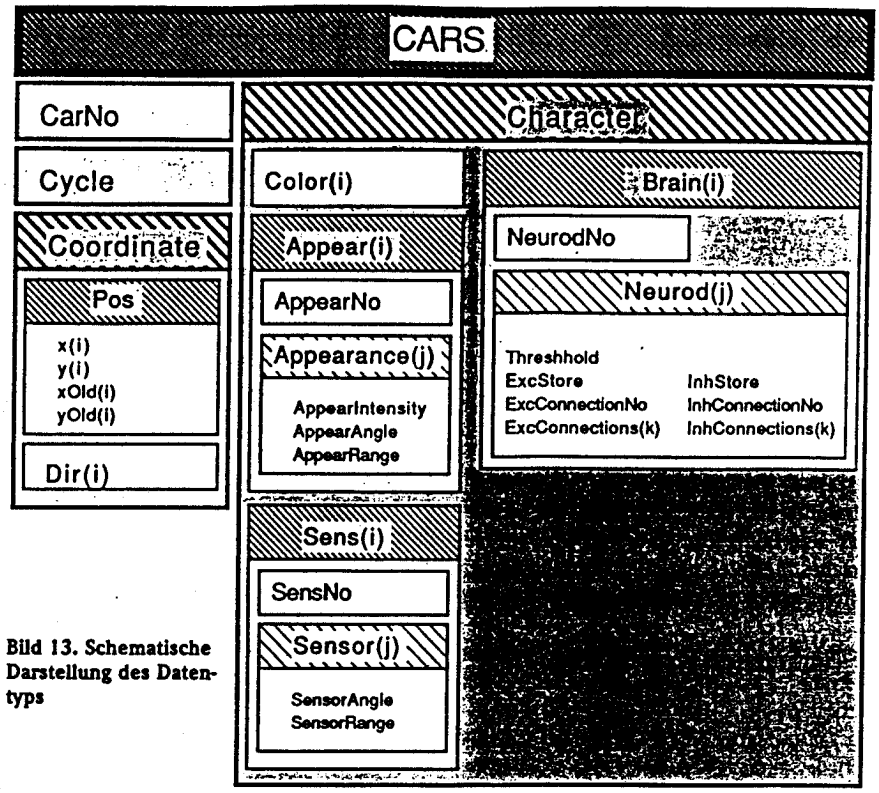


Bild 13. Schematische Darstellung des Datentyps

Das Programm erlaubt es, sich eine eigene Simulationswelt zu konstruieren, in der mehrere Vehikel vorkommen können. Wer über einen schnellen Rechner verfügt (PC-AT, Atari ST, Amiga, usw.), wird sicherlich von der Dynamik begeistert sein, mit der die Vehikel auf dem Bildschirm hin- und herhuschen. Wir möchten allerdings darauf hinweisen, daß bei diesem Vielkörperproblem (jedes Vehikel kann auf alle anderen reagieren) die Rechengeschwindigkeit quadratisch mit der Anzahl der Fahrzeuge abnimmt.

Das abgedruckte Programm bietet im Menü drei Aktionsmöglichkeiten an: die Eingabe von „a" bewirkt den Abbruch des Programms, „w" das Weiterrechnen einer Simulation und „p" aktiviert die Positions- und Richtungseingabe für jedes Fahrzeug. Wurde das Programm gerade gestartet, muß man hier die Positionseingabe auswählen, um die Anfangspunkte der Simulation festzulegen.

Die Welt, in der sich die Vehikel bewegen können, hat ganzzahlige Koordinaten mit einem Wertebereich von −32 000 bis +32 000. Es empfiehlt sich allerdings, Positionen zu wählen, die innerhalb des Bildschirmbereiches liegen. Auf dem Amiga sollte also die x-Koordinate Werte zwischen 0 und 320 und die y-Koordinate Werte zwischen 0 und 240 annehmen. Nach der Positionseingabe wird die Startrichtung abgefragt. Hier muß ein Winkel zwischen 0 und 360 Grad eingegeben werden. Zum Abschluß wird die Anzahl der Simulationsschritte abgefragt. Hier sollte man zunächst Werte um 1000 probieren, um ein Gefühl für die typischen Rechenzeiten zu gewinnen. Als Output liefert das Programm dann eine Strichgrafik der Fahrzeugbewegung.

Am Ende der Berechnung wird oben links auf dem Bildschirm ein kleines Rechteck angezeigt. Wenn nun RETURN betätigt wird, kehrt das Programm ins Menü zurück. Mit „w" kann man jetzt die gerade fertiggerechnete Simulation weiterlaufenlassen. Die Anzahl der simulierten Vehikel und deren genaue Definition muß übrigens im Quellprogramm in der Prozedur DefineCars vorgenommen werden. Dies ist zwar etwas umständlich, eine komfortablere Eingaberoutine hätte jedoch den Rahmen des ohnehin nicht kurzen Programms gesprengt.

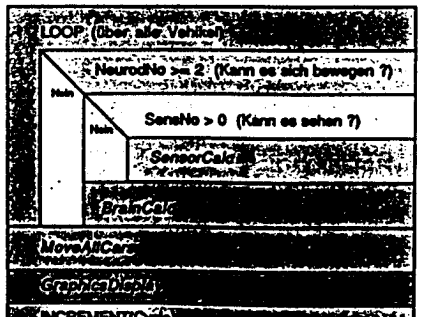


Bild 14. Blockschema des Programm-Hauptteils

Das Programm ist in zwei Module unterteilt; im ersten Modul wird der Datentyp definiert, und im zweiten Modul sind die Berechnungsroutinen untergebracht. Beide Module sind ausführlich kommentiert, so daß an dieser Stelle nur ein Überblick über die allgemeine Funktionsweise des Programms gegeben wird.

Bild 14 zeigt ein Blockschema des Programmablaufes. In einer Schleife werden alle Vehikel durchgerechnet. Zuerst wird getestet, ob das zu berechnende Fahrzeug mindestens zwei Neuroden hat. Wie in Bild 12 (und *Bild 16* unten) ersichtlich, sind die beiden letzten Neuroden immer direkt an die Motoren gekoppelt. Eine Bewegungsberechnung hat also nur bei mindestens zwei Neuroden einen Sinn. Als nächstes wird abgefragt, ob Sensoren vorhanden sind. Wenn ja, wird in der Prozedur SensorCalc die Berechnung der Impulse von den Sensoren durchgeführt, andernfalls kann man sich den Aufruf der Prozedur sparen.

Danach werden in der Prozedur BrainCalc die Neurodenimpulse um einen Takt weitergerechnet und die Motorimpulse, d. h. die Impulse der letzten beiden Neuroden, ermittelt. Wenn die Anweisungen für alle Vehikel durchgeführt sind, werden in der Prozedur MoveAllCars alle Fahrzeuge entsprechend ihren Motorimpulsen bewegt. Dies wird in der Prozedur GraphicsDisplay

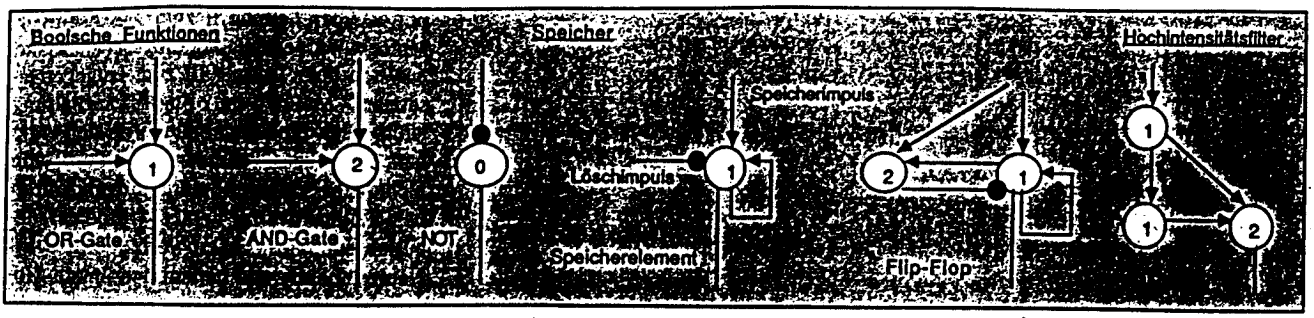


Bild 15. Einfache neurodale Bauelemente

auf dem Bildschirm dargestellt. Abschließend wird der Systemtakt um eins erhöht und wieder von vorne begonnen.

Ein wichtiger Teil des Programms ist die Intensitätsumrechnung. Die Neurodenverschaltung überträgt nur Impulse von 1 oder 0. Die Intensität einer Lichtquelle ist aber im Datentyp als Integer festgelegt, kann also prinzipiell Werte von 0 bis ca. 32 000 annehmen.

In der Nähe einer hellen Lampe erhält der Sensor hohe Werte. Dies bedeutet, daß in aufeinanderfolgenden Takten häufig Eins an das Neurodennetz übergeben wird. Bei schwach einfallender Intensität sendet der Sensor dagegen häufiger Null als Eins. Diese Übersetzung von analogen Intensitätswerten in digitale Impulse haben wir im Programm mit einem Umrechnungsfeld realisiert. Die willkürlich festgelegte maximale Intensität von 10000 entspricht dabei 100% und erzeugt eine Impulsfolge 1111... Die Hälfte dieser Intensität generiert die Folge 101010... .

Das abgedruckte Braitenberg-Programm wurde auf einem Commodore-Amiga mit dem M2-Amiga SDS Modula-2 [3] entwickelt. In einer solchen Umgebung müßte es direkt laufen. Auch auf anderen Rechnern sollten keine größeren Schwierigkeiten auftauchen, da nur wenige Amiga-spezifische Befehle benutzt werden und diese hauptsächlich in den Grafikroutinen auftreten. Die Kommentierung erleichtert die Übertragung auf andere Rechner. An dieser Stelle sei noch angemerkt, daß das ungekürzte Programm auf Diskette verfügbar ist [4], und zwar in einer Version für den Amiga und in einer Version für IBM-PC (Logitech LGT 100 V 3.0). Diese Versionen sind umfangreicher und wesentlich komfortabler als das hier vorgestellte Programm.

Das abgedruckte Programm ist in Modula-2 geschrieben; wirklich Modula-spezifische Tricks waren jedoch nicht notwendig, da alle Algorithmen recht einfacher Natur sind. Dies ist besonders für die Besitzer von Pascal-Compilern von Vorteil, da das Programm deswegen leicht in Pascal umgewandelt werden kann. Hierzu muß das Definitionsmodul BraitenbergDataTypes komplett in das Hauptprogramm eingefügt werden (die entsprechenden IMPORT-Anweisungen sind dann natürlich überflüssig).

Außerdem ist bei der Übertragung in Pascal zu beachten: die Prozeduren AngleRange-Test, LightFromOneSource und LightOnOneSensor müssen in Pascal als FUNCTION deklariert werden; die LOOP-Anweisung im Hauptprogramm muß durch eine geeignete andere Schleifenanweisung (z.B. WHILE DO) ersetzt werden. In Modula-2 wird ja eine LOOP-Schleife so lange durchlaufen, bis eine EXIT-Anweisung angetroffen wird. Es sollte darauf geachtet werden, die Pascal-spezifischen BEGINs bei Schleifen und WITH-DO-Anweisungen einzufügen, die in Modula-2 nicht mehr zugelassen sind.

Das Braitenberg-Programm stellt im Grunde eine Programmiersprache zur Verfügung, in der Simulationen definiert werden können. Ein einfaches Beispiel hierfür ist die Programmierung einer Lichtquelle und eines Aggressors. Wie die Prozedur DefineCars zeigt, wird eine Lichtquelle folgendermaßen definiert: unter AppearAngle wird die Strahlungsrichtung angegeben, und unter AppearRange der abgedeckte Winkelbereich. Zum Beispiel decken AppearAngle=45 und AppearRange=50 den Bereich von -5 Grad (45-50) bis 95 Grad (45+50) ab. Die Festlegung der Sensoren mit SensorAngle und SensorRange funktioniert nach dem gleichen Schema.

Bei einigen Simulationen sollte darauf geachtet werden, daß die Sensoren zusammen den gesamten Winkelbereich von 360 Grad abdecken – sonst kann es passieren, daß die Vehikel plötzlich stehen bleiben, weil sie nichts mehr sehen. Bei der Programmierung der Neurodenverschaltung ist wichtig, daß bei jedem Neurod lediglich angegeben wird, woher es Informationen bezieht, d. h. nur die einlaufenden Impulsleitungen werden jeweils aufgezählt. Exzitatorische und inhibitorische Verbindungen werden in separaten Feldern aufgeführt.

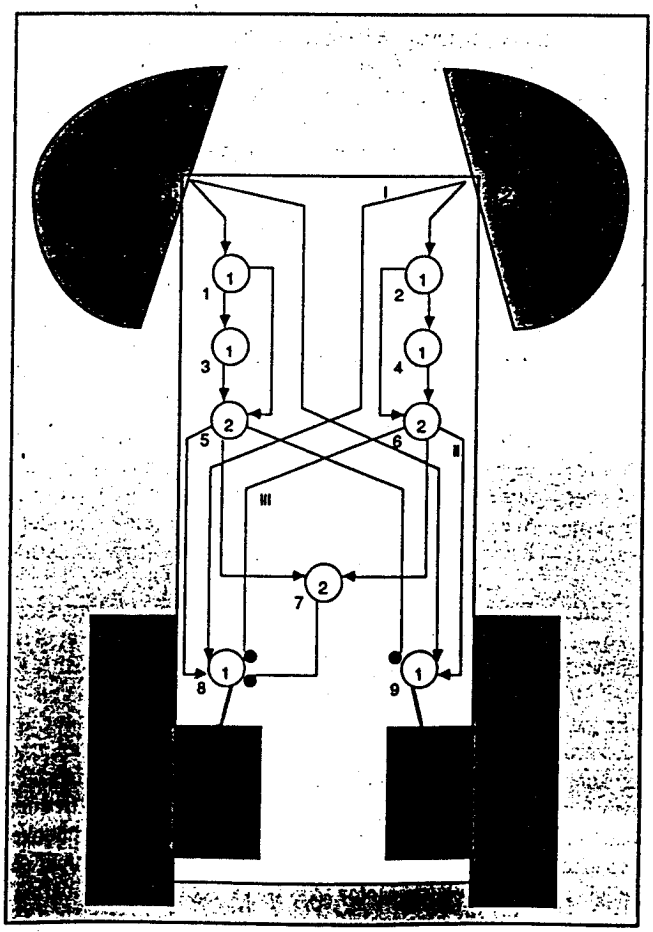


Bild 16. Neurodenverschaltung des Haßliebe-Vehikels

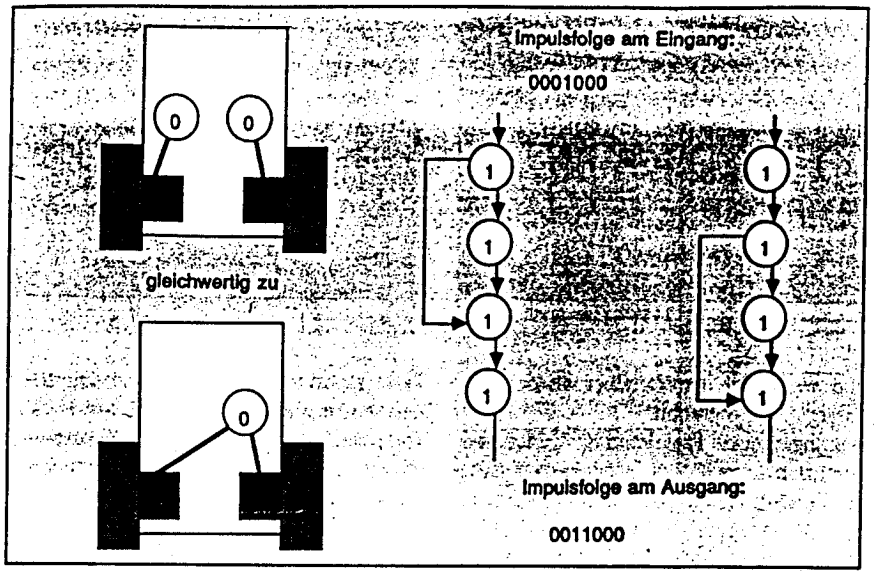


Bild 17. Konstruktionsalternativen in der Neurodenlogik

Vor der Definition eines Vehikels sollte man dessen Neuroden und Sensoren durchnumerieren, dabei muß beachtet werden, daß das letzte Neurod immer an den rechten Motor gekoppelt ist und das vorletzte Neurod an den linken Motor (siehe Bild 16 unten). Alle Verbindungen zwischen den Neuroden sind durch die Neurodennummern charakterisiert. Negative Verbindungsnummern entsprechen Verbindungen zu den Sensoren, also bedeutet z. B. eine Verbindung zu −1 eine Verbindung zum ersten Sensor.

Das Neurodenkonzept mit seiner Flexibilität durch Schwellwerte und Parallelverarbeitung eröffnet verschiedenste Programmiermöglichkeiten. Einige interessante elementare Verschaltungen sind in *Bild 15* dargestellt. Wie man sieht, können Boolsche Vergleiche (AND, OR) und eine Negierung (NOT) leicht durch ein Neurod realisiert werden. Ebenfalls mit nur einem Neurod wird ein einfaches Speicherelement ermöglicht: ein Impuls (entspricht einer Eins) bleibt so lange erhalten, bis ein Impuls über die Löschleitung kommt. Mit zwei Neuroden kann ein Flip-Flop konstruiert werden: ein einlaufender Impuls bleibt solange erhalten, bis ein zweiter kommt, der dann automatisch als Löschimpuls wirkt.

Interessant ist schließlich auch der Entwurf eines Hochintensitätsfilters (HIF), der ein nichtlineares Verhalten verursacht. Er gibt Impulse nur dann weiter, wenn mindestens zwei Impulse direkt hintereinander kommen. In der Praxis bedeutet dies, daß der HIF erst bei Intensitäten über 50% langsam aktiv wird. Höhere Intensitäten lassen sich ebenfalls abfragen, man muß dann lediglich mehr Neuroden mit Schwellwert 1 vorschalten und den Schwellwert 2 entsprechend erhöhen. Mit Kombinationen der in Bild 15 gezeigten Logik- und Speicherelemente können natürlich auch weitaus kompliziertere Funktionsgruppen zusammengesetzt werden, z. B. Taktgeber, Addierer, Stoppuhren, usw.

Die Programmierung eines etwas komplexeren Vehikels kann am Beispiel des Haßliebe-Fahrzeugs illustriert werden, dessen Verschaltung in Bild 16 dargestellt ist. Dieses Fahrzeug vereinigt die Schaltungen von Feigling und Aggressor (Bild 12). Die Entscheidung darüber, welcher der beiden Typen jeweils aktiv ist, trifft ein HIF. Bei hoher Intensität (also in der Nähe von Lichtquellen) verhält es sich wie ein Feigling, bei geringer Intensität (also größerer Entfernung von den Lampen) wie ein Aggressor. Die Bahnkurve des Haßliebe-Wesens ist in ihrem Anfangsstadium in *Bild 3* gezeigt. Wenn man das Fahrzeug länger fahren läßt, so ensteht eine recht chaotische Bahn, die als rote Linie in *Bild 4* dargestell ist und die an eine Blume erinnert. In Bild 4 fahren zusätzlich drei Liebhaber-Vehikel (grün) von unten auf die Lichtquelle zu und bleiben kurz vor ihr stehen. Mit ihrer Bahn erzeugen sie den Eindruck eines Blumenstiels.

Wie funktioniert die in Bild 16 dargestellte Neurodenverschaltung des Haßliebe-Fahrzeugs? Das Vehikel ist fast symmetrisch, so daß wir uns zunächst auf die Diskussion der rechten Seite beschränken können. Die Neuroden 2, 4, 6 bilden den bereits bekannten HIF, d.h. sie lassen nur bei hoher Intensität Impulse passieren. Bei geringer Intensität ist also nur Leitung I zum entgegengesetzten Motor aktiv – es ergibt sich eine Aggressorschaltung. Anders bei hoher Intensität. Der HIF wird aktiv und mit Impulsen über Leitung II zum gleichseitigen Motor wird der Feigling eingeschaltet und zugleich über Leitung III zum entgegengesetzten Motor die Aggressorschaltung lahmgelegt. Ein Sonderfall würde sich ergeben, wenn das Fahrzeug direkt auf eine Lichtquelle zufahren würde; dann wären beide HIF aktiv, das Vehikel würde geradeausfahren. Um das zu verhindern, wird Neurod 7 aktiv und hemmt den linken Motor: das Fahrzeug dreht ab.

Wie wird das Vehikel nun programmiert? Wiederum müssen Sensoren und Neuroden durchnumeriert werden. Für jedes Neurod muß dann angegeben werden, wieviele einlaufende exzitatorische und inhibitorische Verbindungen vorhanden sind und woher sie kommen. Dieses Fahrzeug reagiert übrigens recht empfindlich auf Veränderungen des Sensorbereiches. Für das gezeigte Beispiel des Haßliebe-Fahrzeuges ist das Quellprogramm der verwendeten Sensorkonfiguration und der Neurodenverschaltung in *Bild 19* aufgeführt.

Das Braitenberg-Programm ermöglicht schon in der abgedruckten Kurzfassung interessante Simulationen. Alle bereits vorgestellten Situationen mit mehreren Vehikeln lassen sich nachspielen, und natürlich viele andere, die sich z. B. durch Variation der Parameter (Sensoren usw.) ergeben. Schon bei wenigen Vehikeln ergeben sich chaotische Bahnen, die extrem empfindlich auf unterschiedliche Anfangsbedingungen reagieren.

Dies wird in *Bild 5* besonders deutlich, das eine kreisende, punktförmige Lichtquelle („Rotator") darstellt, welche drei Aggressoren anlockt, die mit fast gleichen Anfangsbedingungen losfahren. Die Aggressoren haben identische Sensorbereiche und Neurodenverschaltungen, sind aber im gezeigten Bild der Übersichtlichkeit wegen farblich unterschiedlich gekennzeichnet. Beim Start weicht die Ausrichtung des zweiten um nur ein Grad von der des ersten ab, während der dritte Aggressor in seiner x-Position um eine Einheit gegenüber den andern beiden verschoben ist. Schon nach kurzer Zeit laufen die Bahnkurven vollkommen auseinander.

Bei diesem Beispiel tragen die Aggressoren selbst keine Lichtquellen, sie merken also nichts voneinander und sehen nur den Rotator. Dieser ist blind, hat also keine Sensoren, und fährt immer ein kurzes Stück geradeaus, mit einer anschließenden kleinen Drehung nach links. Auf die konkrete Neurodenverschaltung des Rotators sei hier nicht eingegangen, sie ist jedoch nicht übermäßig kompliziert.

Als Anregung an den Leser, sich selbst an Braitenberg-Fahrzeugen zu versuchen, sind

```
DEFINITION MODULE BraitenbergDataTypes;
(*****************************************************************************)
(* In diesem Modul wird der Datentyp definiert, den wir benutzen, um die     *)
(* Braitenberg Vehikel zu simulieren. Dieses Programm läuft ohne Änderungen  *)
(* auf einem Commodore Amiga mit M2-Amiga Modula [V. 3.1d]. Beim Übertragen  *)
(* auf andere Computer achte man bitte auf die Kommentare.                   *)
(*****************************************************************************)
(* (c) Feb. 1988 by Klaus Boehncke, Christoph Köhler, Georg Meyer-Berg       *)
(*          Permission granted to freely distribute with this notice !       *)
(*****************************************************************************)
(* Program Version-Amiga mc/1.3, developed using M2-Amiga Software Deve-     *)
(* lopment System by A.+L. Meier-Vogt, Im Späten 23, CH-8906 Bonstetten.     *)
(*****************************************************************************)

(* Das M2SDS-Modula auf dem Amiga exportiert automatisch ALLES im Definitionsmodul.
   Bei anderen Modula-Implementationen muß an dieser Stelle eventuell eine Ex-
   portliste vereinbart werden (Siehe Modul Braitenberg für eine Auflistung
   aller benötigten Typen). Alternativ kann dieses Definitionsmodul auch kom-
   plett in das Hauptmodul übernommen werden. *)

CONST  MaxCarNo = 20;                    (* Maximalzahl der Vehikel pro Simulation *)
       MaxSenseNo = 3;                   (* Maximalzahl der Sensoren pro Vehikel *)
       MaxAppearNo = 3;                  (* Maximalzahl der Lichtquellen pro Vehikel *)
       MaxNeurodNo = 20;                 (* Maximalzahl der Neuroden pro Vehikel *)
       MaxConnectionNo = MaxNeurodNo DIV 3;
                                         (* Maximalzahl der Verbindungen pro Neurod *)

TYPE   AppearNoType = [0..MaxAppearNo];  (* Nummerierung der Lichtquellen *)
       NeurodNoType = [-MaxSenseNo..MaxNeurodNo]; (* Nummerierung der Neuroden,
                                         negative Werte verbinden zu den Sensoren *)
       SenseNoType = [0..MaxSenseNo];    (* Nummerierung der Sensoren *)
       CarNoType = [0..MaxCarNo];        (* Nummerierung der Vehikel *)
       ConnectionNoType = [0..MaxConnectionNo];(* Nummerierung der Verbindun-
                                                  gen *)
       (*-------------------------------------------------------------------- *)

       NeurodType =                      (* Beschreibung eines Neurods *)
         RECORD
           Threshho:d: INTEGER;          (* Schwellwert des Neurods *)
           InhStore, ExcStore: INTEGER;  (* Speichern den Neurodenstatus *)
           ExcConnectionNo: ConnectionNoType; (* Anzahl der exz. Verbindungen *)
           ExcConnections: ARRAY ConnectionNoType OF NeurodNoType;
                                         (* Feld enthält Verbindungen *)
           InhConnectionNo: ConnectionNoType; (* Anzahl der inh. Verbindungen *)
           InhConnections: ARRAY ConnectionNoType OF NeurodNoType;
         END;                            (* Feld enthält Verbindungen *)

       CarBrainType =                    (* Beschreibung des "Gehirns" eines Vehikels *)
         RECORD
           NeurodNo: NeurodNoType;       (* Anzahl der Neuroden *)
           Neurod: ARRAY NeurodNoType OF NeurodType;
         END;                            (* Feld enthält Neuroden *)

       BrainType = ARRAY CarNoType OF CarBrainType;
                   (* Beschreibung des Feldes mit den "Gehirnen" aller Vehikel *)

       (*-------------------------------------------------------------------- *)

       SensorType =                      (* Beschreibung eines Sensors *)
         RECORD
           SensorAngle: INTEGER;         (* Ausrichtung in Grad [0..360] *)
           SensorRange: INTEGER; (* Abgedeckter Winkelbereich rechts u. links *)
         END;                            (* von der Ausrichtungsorientierung  [..180] *)

       CarSensType =                     (* Beschreibung der Sensoren eines Vehikels *)
         RECORD
           SensNo: SensNoType;           (* Anzahl der Sensoren *)
           Sensor: ARRAY SenseNoType OF SensorType;
         END;                            (* Feld enthält alle Sensoren eines Vehikels *)

       SenseType = ARRAY CarNoType OF CarSensType;
                   (* Beschreibung des Feldes mit den Sensoren aller Vehikel *)
```

Bild 18. Ein Programm zur Simulation von künstlichen Wesen (in Modula-2 auf dem Amiga)

```
       (*-------------------------------------------------------------------- *)

       AppearanceType =      (* Beschreibung einer Erscheinung (Lichtquelle) *)
         RECORD
           AppearIntensity: INTEGER;     (* Stärke der Lichtquelle *)
           AppearAngle: INTEGER;         (* Ausrichtung in Grad [0..360] *)
           AppearRange: INTEGER; (* Abgedeckter Winkelbereich linkau rechts *)
         END;

       CarAppearType =       (* Beschreibung aller Lichtquellen eines Vehikels *)
         RECORD
           AppearNo: AppearNoType;       (* Anzahl der Lichtquellen *)
           Appearance: ARRAY AppearNoType OF AppearanceType;
         END;                (* Feld enthält alle Lichtquellen eines Vehikels *)

       AppearType = ARRAY CarNoType OF CarAppearType;
              (* Beschreibung des Feldes mit den Lichtquellen aller Vehikel *)

       (*-------------------------------------------------------------------- *)

       CharacterType =       (* Beschreibung der gesamten Eigenschaften aller
                                Vehikel *)
         RECORD
           Color:ARRAY CarNoType OF CARDINAL; (*Feld mit Farbcodes der Vehikel
                                               für die Graphikdarstellung *)
           Appear: AppearType;                (* Lichtquellen *)
           Sens: SensType;                    (* Sensoren *)
           Brain: BrainType;                  (* Gehirn = Neuronenverschaltung *)
         END;

       (*-------------------------------------------------------------------- *)

       DirectionType = ARRAY CarNoType OF INTEGER; (* Beschreibung des Feldes
                                mit der Ausrichtung (= Orientierung)
                                jedes Vehikels in Grad [0..360] *)

       PositionType =              (* Beschreibung der Positionen aller Vehikel *)
         RECORD
           x,y,xOld,yOld : ARRAY CarNoType OF INTEGER;
         END;                  (* Feld mit alter u. neuer Position aller Vehikel *)

       CoordinateType =          (* Beschreibung der Orientierung und Position
                                    aller Vehikel *)
         RECORD
           Pos: PositionType;                 (* Position *)
           Dir: DirectionType;                (* Ausrichtung *)
         END;

       (*-------------------------------------------------------------------- *)

       CarsType =                        (* Enthält alle Daten und Vehikel *)
         RECORD
           CarNo: CarNoType;             (* Anzahl der Vehikel *)
           Cycle: CARDINAL;      (* Simulationstakt, läuft im Bereich [1..100] *)
           Character: CharacterType;     (* Eigenschaften der Vehikel *)
           Coordinate: CoordinateType; (* Orientierung & Position der Vehikel *)
         END;

       (*-------------------------------------------------------------------- *)

END BraitenbergDataTypes.

IMPLEMENTATION MODULE BraitenbergDataTypes;

BEGIN
END BraitenbergDataTypes.

MODULE Braitenberg;

(*****************************************************************************)
(*  Dieses Modul ist das Braitenberg-Hauptprogramm. Die Vehikel werden durch *)
(*  simuliert und deren Bewegung als Strichgrafik auf dem Bildschirm gezeigt.*)
(*****************************************************************************)
```

```
(*  (c) Feb 1988 by Klaus Boehncke, Christoph Köhler, Georg Meyer-Berg     *)
(*      Permission granted to freely distribute with this notice !         *)
(*************************************************************************)
(*  Programmversion Amigainc/1.3, developed using M2-Amiga Software Deve-  *)
(*  lopment System by M.-L. Meier-Vogt, Im Späten 23, CH-8906 Bonstetten.  *)
(*************************************************************************)

(* Die folgenden Anweisungen sind Amiga-spezifisch [Grafik u.ä.]: *)
  FROM SYSTEM    IMPORT ADR, ADDRESS, INLINE;
  FROM Arts      IMPORT Assert, TermProcedure;
  FROM Graphics  IMPORT RastPortPtr, ViewModes, ViewModeSet,
                        LoadRGB4, Move, Draw, SetAPen;
  FROM Intuition IMPORT customScreen, stdScreenHeight, NewScreen,
                        ScreenFlags, ScreenFlagSet, ScreenPtr, CloseScreen,
                        OpenScreen;
(* Ende der Amiga-spezifischen Anweisungen. *)

FROM MathLibo    IMPORT sin, cos, arctan;
FROM InOut       IMPORT WriteCard, ReadInt, ReadCard, WriteString,
                        WriteInt, WriteLn, ReadString;
                                          (* Ein/Ausgabeanweisungen *)

FROM BraitenbergDataTypes IMPORT CarsType,  (* Definiert Daten aller Vehikel,
                                    siehe auch Modul BraitenbergDataTypes *)
                                 MaxNeurodNo,(* Maximale Anzahl der: Neuroden *)
                                 MaxCarNo,   (*                      Vehikel *)
                                 MaxSenseNo; (*                      Sensoren *)


CONST  RadToDeg= 57.295779513;       (* Konvertierungsfaktor Radiant --> Grad *)
       DegToRad= 0.0174532925;       (* Konvertierungsfaktor Grad --> Radiant *)

TYPE   ImpulseType=[0..1];                    (* Impuls ist entweder 1 oder 0 *)
       MotorImpulsesType=ARRAY[1..MaxCarNo] OF BOOLEAN;
                                  (* Feld um die Impulse an die Motoren zu speichern *)
       SensorImpulsesType=ARRAY[1..MaxSenseNo] OF ImpulseType;
                                  (* Feld um die Impulse von den Sensoren zu speichern *)
       IntensityType=ARRAY[0..100],[1..100] OF ImpulseType;
                                  (* Feld für die Intensitätsumrechnung - siehe Kommentar zur Prozedur
                                                                        "Initialize" *)
VAR    Cars: CarsType;                    (* enthält die Daten aller Fahrzeuge *)
       drawRP: RastPortPtr;               (* speziell für Amiga-Grafik *)
       screen: ScreenPtr;                 (* speziell für Amiga-Grafik *)

(*-------------------------------------------------------------------------*)

(* Im folgenden Abschnitt findet man einige Amiga-spezifische Prozeduren, die
   hauptsächlich die Grafik betreffen. Sie sind teilweise aus dem Demonstra-
   tionsmodul 'Sparks' des M2Amiga-Development Systems übernommen.       *)

PROCEDURE CleanUp;
BEGIN
  IF screen#NIL THEN                  (* Wenn Grafikbildschirm noch nicht geschlossen.. *)
    CloseScreen(screen);                          (* ..dann schließe Grafikbildschirm *)
    screen:=NIL;
  END;
END CleanUp;


PROCEDURE Colors; (*$E-*)                 (* Definiere Farbtabelle des neuen Bildschirms *)
BEGIN
  INLINE
    (0000H,0FFFH,0F00H,0D3FAH,0E3OH,0FB0H,0FF2H,0BF0H,
     05D0H,000FH,036FH,077FH,0C0EH,0F2EH,0ABCH,0DB8H);
    (* Diese Hexzahlen geben Farbwerte an. Die 2.,3.,4. Stelle definieren
       jeweils Rot-, Grün- und Blauanteil. Der Einzelwert geht von 0 bis F.
       Z.B. entspricht 0000H schwarz, 0FFFH ist weiß und 0F00H rot. *)
END Colors;

PROCEDURE CreateScreen (w,h,d: INTEGER; t: ADDRESS): ScreenPtr;
VAR ns: NewScreen;                        (* Eröffne neuen Grafikbildschirm *)
BEGIN
  WITH ns DO
    leftEdge:=0; topEdge:=0; width:=w; height:= h; depth:=d;
    detailPen:=0; blockPen:=1;
    viewModes:=ViewModeSet{};
    type:=customScreen;
    font:=NIL;
    defaultTitle:=t;
    gadgets:=NIL;
    customBitMap:=NIL;
  END;
  RETURN OpenScreen(ns);
END CreateScreen;

(* Ende der Amiga-spezifischen Prozeduren ! *)

(*-------------------------------------------------------------------------*)

(* In dieser Prozedur können die Vehikel definiert werden, die bei der Simula-
   tion benutzt werden. Als Beispiel ist hier eine Lampe (Vehikel 1) und ein
   Agressor (Vehikel 2) definiert. *)

PROCEDURE DefineCars;

BEGIN
  WITH Cars DO
   (* Ab hier können eigene Simulationswelten definiert werden: *)
   CarNo:=2;   (* Anzahl der Vehikel *)
    (* Erstes Vehikel: Lichtquelle *)
    WITH Character DO
      WITH Appear[1] DO                          (* Licht des ersten Vehikels *)
        AppearNo:=1;                                    (* Nur eine Lichtquelle *)
        Appearance[1].AppearIntensity:= 32000;          (* Fast max. Intensität *)
        Appearance[1].AppearAngle:= 0;       (* Strahlt in Richtung 0 Grad... *)
        Appearance[1].AppearRange:= 180;  (* Bereich bis +- 180, d.h. rundum *)
      END;
      (* Zweites Vehikel: Aggressor *)
      WITH Sens[2] DO                        (* Sensoren des zweiten Vehikels *)
        SensNo:=2;                                      (* Zwei Sensoren *)
        Sensor[1].SensorAngle:=80;     (* Linker Sensor 'guckt' nach 80 Grad *)
        Sensor[1].SensorRange:=90;          (* Sichtbereich bis +- 90 Grad *)
        Sensor[2].SensorAngle:=-80;   (* Rechter Sensor 'guckt' nach 280 Grad *)
        Sensor[2].SensorRange:=90;          (* Sichtbereich bis +- 90 Grad *)
      END;
      WITH Brain[2] DO                       (* 'Gehirn' des zweiten Vehikels *)
        NeurodNo:=2;                                    (* Zwei Neuroden *)
        Neurod[1].Threshhold:=1; (* Schwellwert des 1. Neurods = linker Motor *)
        Neurod[1].ExcConnectionNo:=1;   (* Eine exzit. Verbindung, und zwar...*)
        Neurod[1].ExcConnections[1]:=-2;          (* ... zum rechten Sensor *)
            (* würde dort eine -1 stehen, d.h. linker Motor an linken
               Sensor, und unten beim 2ten Neurod eine -2, d.h. rechter
               Motor an rechten Sensor, dann würde ein 'Feigling' definiert! *)
        Neurod[1].InhConnectionNo:=0;          (* Keine inh. Verbindungen *)
        Neurod[2].Threshhold:=1;(* Schwellwert des 2. Neurods = rechter Motor *)
        Neurod[2].ExcConnectionNo:=1;   (* Eine exzit. Verbindung, und zwar...*)
        Neurod[2].ExcConnections[1]:=-1;          (* ... zum linken Sensor *)
        Neurod[2].InhConnectionNo:=0;          (* Keine inh. Verbindungen *)
            (* Wenn man die inhibitorischen Vehikel - also Liebhaber und
               Flüchtling - definieren möchte, so muß man lediglich statt der
               ExcCon... die InhCon... definieren und ExcConnectionNo auf 0
               setzen. Die Schwellwerte müßen dann auf 0 gesetzt werden.  *)
      END;
      Color[1]:=1; (* Farbcode für die Graphikdarstellung, erstes Vehikel ... *)
      Color[2]:=2; (* .. hat Farbe Nr. 1, das zweite Vehikel Farbe Nr. 2.  *)
    END;
  END;
END DefineCars;

(*-------------------------------------------------------------------------*)

(* Die folgende Prozedur erlaubt die Positions- und Richtungseingabe für alle
   simulierten Vehikel *)

PROCEDURE CoordinateInput;

VAR   CarIndex: INTEGER;
BEGIN                                          (* Index des Vehikels *)
  WITH Cars.Coordinate.Pos DO
    FOR CarIndex:= 1 TO Cars.CarNo DO          (* Schleife über alle Vehikel *)
      WriteLn;
```

```
    WriteString("Positions- und Richtungseingabe / Auto Nr.");
    WriteInt(CarIndex,2);
    WriteLn;
    WriteString("x-Position: ");
    ReadInt(x[CarIndex]);                          (* Einlesen x-Koordinate *)
    WriteString("y-Position: ");
    ReadInt(y[CarIndex]);                          (* Einlesen y-Koordinate *)
    WriteString("Anfangsrichtung: ");
    ReadInt(Cars.Coordinate.Dir[CarIndex]);        (* Einlesen der Richtung *)
    xOld[CarIndex]:=x[CarIndex];  (* Setze alte Koordinate auf Anfangswerte *)
    yOld[CarIndex]:=y[CarIndex];
  END;
 END;
END CoordinateInput;

(*-------------------------------------------------------------------------*)

(* Die folgende Prozedur initialisiert das Feld "Intensity", das für die In-
   tensitätsumrechnung benötigt werd. Intensitätsabhängig werden Impulse auf
   die 100 Taktzyklen verteilt. So bedeutet Intensität 0 eine Folge von 100
   mal "0", Intensität 75% eine möglichst gleichmäßige Verteilung von 75 1ern
   auf 100 Zyklen (entspricht 11101110...). Das Feld "Intensity" hat als erste
   (x) Koordinate die einfallende Intensität (Integer [0..100]) in Prozent und
   als zweite (y) Koordinate den Systemtakt (Integer [1..100]). "Intensity"
   ist also ein 101 x 100 Feld.                                            *)

PROCEDURE Initialize (VAR Intensity: IntensityType);

 VAR IntensityIndex, CycleCounter: CARDINAL;    (* x und y Koord. des Feldes *)
     Quotient, Counter: REAL;                              (* Wird benutzt, s.u. *)
     TestValue: CARDINAL;                                  (* Wird benutzt, s.u. *)

BEGIN
 WriteString("Bitte warten! Programm wird initialisiert: ");
 WriteLn;
 WriteLn;
 FOR IntensityIndex:=0 TO 100 DO                          (* Intensität in Prozent *)
  WriteString(".");
  Quotient:=FLOAT(IntensityIndex)/100.0; (* Prozent--> Bruch,z.B. 50%=0.5 *)
  Counter:=0.0;
  TestValue:=1;                          (*Vergleichswert [ganze Zahl], fängt bei 1 an *)
  FOR CycleCounter:=100 TO 1 BY -1 DO                     (* Taktzyklus *)
   Counter:=Counter+Quotient;                    (* Summiere Prozentbruch auf *)
   IF ((FLOAT(TestValue)-Counter)<8.0E-4) THEN     (* Wenn die Summe des *)
    Intensity[IntensityIndex,CycleCounter]:= 1;(* Bruches eine ganze Zahl*)
    INC(TestValue);                    (* Übersteigt, muß der Impuls=1 gesetzt werden *)
   ELSE
    Intensity[IntensityIndex,CycleCounter]:= 0;
   END;
  END;                                             (* sonst setze Impuls=0 *)
 END;
END Initialize;

(*-------------------------------------------------------------------------*)

(* In der folgenden Prozedur werden die Sensoren eines Vehikels durchgerechnet
   und alle eventuell darauf einfallenden Intensitäten aufsummiert. Der "ana-
   loge" Prozentwert der Gesamtintensität wird mit Hilfe des oben besprochenen
   Intensitätsfeldes in einen digitalen Sensoroutput gewandelt.           *)

PROCEDURE SensorCalc (VAR Regulation: DigitalType; CurrentCar: INTEGER;
                                                   VAR Intensity: IntensityType);

                    (*CurrentCar: Nummer des Vehikels das gerade berechnet wird *)
                    (*Regulation: Feld das nachher die Digitalimpulse der Sen-
                                                                  soren enthält *)
 VAR Min, Max, OtherCar, CurrentSensor: INTEGER;
     IntensitySum: CARDINAL;
     rSquared: ARRAY [1..MaxCarNo] OF LONGCARD;            (* Abstandsquadrat *)
     Angle: ARRAY [1..MaxCarNo] OF INTEGER;  (* Absolutwinkel zu allen anderen
                                                                    Vehikeln *)
```

```
(* In dieser Unterprozedur werden die Winkel und die Abstands-
   quadrate zu allen anderen Vehikeln berechnet. Die Winkel
   werden im Feld "Angle" und die Abstandsquadrate im Feld
   "rSquared" abgespeichert.                                             *)

PROCEDURE AngleAndDistance (CurrentCar: INTEGER);

    (* CurrentCar: Nummer des Vehikels für das die Berechnung gestartet wird *)

VAR  OtherCar, DiffX, DiffY: INTEGER;
     LongDiffX, LongDiffY: LONGCARD;
BEGIN
  FOR OtherCar := 1 TO Cars.CarNo DO  (* Schleife läuft über alle Vehikel *)
    IF ((OtherCar#CurrentCar) AND    (* Alle anderen Vehikel mit mindest. *)
       (Cars.Character.Appear[OtherCar].AppearNo>=1))THEN (*einer Licht- *)
      WITH Cars.Coordinate.Pos DO                                  (*quelle*)
        DiffX:=x[OtherCar]-x[CurrentCar];          (* Differenz x-Koordinate *)
        DiffY:=y[CurrentCar]-y[OtherCar];          (* Differenz y-Koordinate *)
        LongDiffX:=LONGCARD(ABS(DiffX));
        LongDiffY:=LONGCARD(ABS(DiffY));
        rSquared[OtherCar]:= LongDiffX*LongDiffX+LongDiffY*LongDiffY;
        (* Anfang Winkelberechnung *)
        IF DiffX#0 THEN
          Angle[OtherCar]:=TRUNC(arctan(FLOAT(DiffY)/FLOAT(DiffX))
                                                                  *RadToDeg);
        ELSIF DiffY>0 THEN
          Angle[OtherCar]:=90;
        ELSE
          Angle[OtherCar]:=270;
        END;
        IF DiffX<0 THEN
          Angle[OtherCar]:=180+Angle[OtherCar];
        END;
        (* Ende Winkelberechnung *)
    END;
  END;
END AngleAndDistance;

(*.....................................................................*)

(* Die nächste Prozedur liefert TRUE, wenn ein Winkel (TestAngle)
   sich zwischen zwei anderen (Min und Max) Winkeln befindet.
   Liegt der Testwinkel außerhalb, so liefert die Prozedur FALSE.        *)

PROCEDURE AngleRangeTest (Min, Max, TestAngle: INTEGER): BOOLEAN;

BEGIN
  Max:=(Max+720) MOD 360;
  Min:=(Min+720) MOD 360;          (* Sicherste Lösung (MOD 360) ist [0..360] *)
  TestAngle:=(TestAngle+720) MOD 360;
  IF (Min<Max) THEN
    RETURN ((Min<=TestAngle) AND (TestAngle<=Max))
  ELSE                                                          (* Abfrage *)
    RETURN ((Min<=TestAngle) OR (TestAngle<=Max));
  END;
END AngleRangeTest;

(*.....................................................................*)

(* Die folgende Prozedur liefert die Intensität, die das betrachtete
   Vehikel (Nr. CurrentCar) von einem anderen Vehikel (Nr. OtherCar)
   mit Lichtquelle erhält. Der Intensitätswert bewegt sich im
   Bereich [0..10000].                                                   *)

PROCEDURE LightFromOneSource (OtherCar, CurrentCar: INTEGER): CARDINAL;
VAR Min, Max, Ang: INTEGER;
    IntensityValue: CARDINAL;
    AppearIndex: INTEGER;
    TemporaryStore: LONGCARD;
```

```
  BEGIN
    IntensityValue:=0;
    WITH Cars.Character.Appear[OtherCar] DO          (* Schleife über alle  *)
      FOR AppearIndex:=1 TO INTEGER(AppearNo) DO     (* Lichtquellen des an-*)
        WITH Appearance[AppearIndex] DO              (* deren Vehikels      *)
          (*Berechnung der min,max Öffnungswinkel der Lichtquelle *)
          Min:=AppearAngle-AppearRange+Cars.Coordinate.Dir[OtherCar];
          Max:=AppearAngle+AppearRange+Cars.Coordinate.Dir[OtherCar];
          Ang:=180+Angle[OtherCar];  (* Winkel vom anderen Vehikel gesehen *)
          (*Jetzt Test, ob das betrachtete Vehikel auch im Konus der
            Lichtquelle sitzt, die vom anderen Vehikel ausgestrahlt wird:  *)
          IF AngleRangeTest(Min,Max,Ang) THEN
            (*Verhindere Programmcrash wenn zwei Vehikel auf gleichem Platz:*)
            IF rSquared[OtherCar]=0 THEN rSquared[OtherCar]:=1; END;
            (*Berechne einfallende Intensität mit quadr. Abstandsgesetz: *)
            TemporaryStore:=LONGCARD(125.0*FLOAT(AppearIntensity)/
                                     FLOAT(rSquared[OtherCar]));
            IF TemporaryStore>10000 THEN          (* 10000 = max. Intensität *)
              IntensityValue:=IntensityValue+10000;
            ELSE
              IntensityValue:=IntensityValue+CARDINAL(TemporaryStore);
            END;
          END;
        END;
      END;
    END;
    RETURN IntensityValue
  END LightFromOneSource;

  (*.....................................................................*)

  (* In der folgenden Prozedur wird lediglich getestet, ob sich eine
     Lichtquelle auf dem anderen Vehikel (OtherCar) im sichtbaren
     Bereich des betrachteten Sensors (CurrentSensor) befindet. Wenn
     ja (so) wird zur Intensitätsberechnung die Prozedur "LightFrom-
     OneSource" aufgerufen.                                          *)

  PROCEDURE LightOnOneSensor (OtherCar, CurrentCar, CurrentSensor: INTEGER)
                                                              : CARDINAL;
  VAR Ang, Min, Max: INTEGER;
  BEGIN
    WITH Cars.Character.Sens[CurrentCar].Sensor[CurrentSensor] DO
      (*Berechnung der min,max Winkel des sichtbaren Bereichs *)
      Min:=SensorAngle-SensorRange+Cars.Coordinate.Dir[CurrentCar];
      Max:=SensorAngle+SensorRange+Cars.Coordinate.Dir[CurrentCar];
    END;
    Ang:=Angle[OtherCar];                        (* Winkel zum anderen Vehikel *)
    IF AngleRangeTest(Min,Max,Ang) THEN (* Test: liegt Ang zwischen Min,Max *)
      RETURN LightFromOneSource(OtherCar,CurrentCar);
    ELSE
      RETURN 0                                   (* Sonst wird nichts gesehen *)
    END;
  END LightOnOneSensor;

  (*.....................................................................*)

BEGIN (*SensorCalc*)
  AngleAndDistance(CurrentCar);
  FOR CurrentSensor:=1 TO Cars.Character.Sens[CurrentCar].SensNo DO
    (* Berechnung der Intensitäten die auf den Sensor fallen *)
    IntensitySum:=0;
    FOR OtherCar:=1 TO Cars.CarNo DO                 (* alle Lichtquellen *)
      IF OtherCar<>CurrentCar THEN
        IntensitySum:=IntensitySum+LightOnOneSensor(OtherCar,
                                   CurrentCar,CurrentSensor);
        (*Summiere alle einfallenden Intensitäten auf *)
        IF IntensitySum>10000 THEN IntensitySum:=10000; END;
      END                                  (* 10000 = Maximale Intensität *)
    END;
    Regulation[CurrentSensor]:=Intensity((IntensitySum DIV 100),Cars.Cycle);
    (*Die analoge Intensität [0..10000] wird mit Hilfe des Feldes "Intensity"
      in einen Digitalimpuls umgewandelt *)
  END;
END SensorCalc;
```

```
(*-------------------------------------------------------------------------*)

(* In der folgenden Prozedur wird das "Gehirn" eines Vehikels, d.h. die Neu-
   rodenverschaltung einen Takt weitergerechnet und die resultierenden Impulse
   an die Motoren ermittelt (sie werden in ImpulseLeft und -Right gespeichert).
   Die Motorimpulse entsprechen dem Output der letzten zwei Neuroden [rechter
   Antrieb entspricht letztem Neurod, linker Antrieb dem vorletzten Neurod.  *)

PROCEDURE BrainCalc (Regulation: DigitalType; CurrentCar: INTEGER;
                     VAR ImpulseLeft, ImpulseRight: AllMotorImpulsesType);

VAR j, i: INTEGER;                                    (* Schleifenindizes *)
    Signal: ARRAY[-MaxSensNo..MaxNeurodNo] OF INTEGER; (* Feld das alle
                  Impulse enthält, negative Indize beziehen sich auf die ent-
                  sprechenden Sensoren, positive direkt auf die Neuroden.  *)
BEGIN
  WITH Cars.Character DO
    FOR i := 1 TO Sens[CurrentCar].SensNo DO  (* Umspulen des Sensoroutputs: *)
      Signal[-i]:=Regulation[i];                (* Regulation[+] --> Signal[-] *)
    END;
    WITH Brain[CurrentCar] DO
      FOR i := 1 TO NeurodNo DO
        Signal[i]:=Neurod[i].ExcStore;          (* Neurodenstatus --> Signal[+] *)
      END;
      FOR i :=1 TO NeurodNo DO                  (* Durchrechnen des Neurodenbaums *)
        WITH Neurod[i] DO
          ExcStore:=0;                          (* Setze Hilfsvariable bzw. Output =0 *)
          FOR j := 1 TO ExcConnectionNo DO
            ExcStore:=ExcStore+Signal[ExcConnections[j]];
          END;                  (* Summiere einlaufende exzitatorische Impulse auf *)
          InhStore:=0;                          (* Setze Hilfsvariable =0 *)
          FOR j := 1 TO InhConnectionNo DO
            InhStore:=InhStore+Signal[InhConnections[j]];
          END;              (* Summiere einlaufende inhibitorische Impulse auf *)
          ExcStore:=ExcStore-InhStore;  (* Subtrahiere inh. Impulse von exzit.*)
          IF (ExcStore<Threshhold) THEN
            ExcStore:=0;                        (* Vergleiche Resultat mit Schwellwert *)
          ELSE                                  (* und setze dementsprechenden Output! *)
            ExcStore:=1;
          END;
        END;
      END;
      ImpulseRight[CurrentCar]:= (Neurod[NeurodNo].ExcStore=1);
      ImpulseLeft[CurrentCar]:= (Neurod[NeurodNo-1].ExcStore=1);
    END;                    (* Setze Motorbefehle: Rechtes Rad = letztes Neurod *)
  END;                                          (* Linkes Rad = vorletztes Neurod *)
END BrainCalc;

(*-------------------------------------------------------------------------*)

(* Die nächste Prozedur bewegt die Vehikel je nach den Impulsen des rechten
   oder linken Antriebs. Bekommt das Fahrzeug einen Impuls auf beide Motoren,
   so wird es vorwärtsbewegt. Bei einem Impuls auf nur einen Antrieb wird
   das Vehikel zunächst gedreht und bewegt sich dann ebenfalls um einen
   (kleineren) Betrag vorwärts.                                          *)

PROCEDURE MoveAllCars (ImpulseLeft, ImpulseRight: AllMotorImpulsesType);

CONST  MovePerImpulse = 6.0;   (* Vorwärtsbewegung bei Impuls auf beide Räder *)
       MovePerTurn = 3.0; (* Vorwärtsbewegung nach der Drehung bei Impuls auf
                                                                   ein Rad *)
       DegreesPerImpulse = 10;       (* Drehung in Grad bei Impuls auf ein Rad *)

VAR    CarIndex: INTEGER;                          (* Index des Vehikels *)

  (*.....................................................................*)

  (* Die folgende Unterprozedur übernimmt die Vorwärtsbewegung des
     Vehikels in Richtung "Theta" und Betrag "Move".                     *)

  PROCEDURE Forward (VAR xOld, yOld, x, y: INTEGER; Move: REAL; Theta: INTEGER);
  BEGIN
    xOld:=x;                                (* Alte Koordinaten abspeichern und ... *)
    yOld:=y;
```

```
    x:=x+INTEGER(Move*cos(FLOAT(Theta)*DegToRad));          (* ... neue setzen *)
    y:=y+INTEGER(-Move*sin(FLOAT(Theta)*DegToRad));
  END Forward;

  (*.....................................................*)

BEGIN (* MoveAllCars *)
  FOR CarIndex:=1 TO Cars.CarNo DO                  (* Schleife über alle Vehikel *)
    WITH Cars.Coordinate DO
      IF (ImpulseLeft[CarIndex] AND ImpulseRight[CarIndex]) THEN  (* Vorwärts *)
        WITH Pos DO
          Forward(xOld[CarIndex],yOld[CarIndex],x[CarIndex],y[CarIndex],
                  MovePerImpulse,Dir[CarIndex]);
        END;
      ELSIF ImpulseRight[CarIndex] THEN (*Impuls auf r.Hinterrad=Linksdrehung *)
        Dir[CarIndex]:=Dir[CarIndex]+DegreesPerImpulse;       (* Linksdrehung *)
        IF Dir[CarIndex]>360 THEN Dir[CarIndex]:=Dir[CarIndex]-360; END;
        WITH Pos DO
          Forward(xOld[CarIndex],yOld[CarIndex],x[CarIndex],y[CarIndex],
            MovePerTurn, Dir[CarIndex]); (* Kleine Vorw.Bewegung nach Drehung *)
        END;
      ELSIF ImpulseLeft[CarIndex] THEN (*Impuls auf l.Hinterrad=Rechtsdrehung *)
        Dir[CarIndex]:=Dir[CarIndex]-DegreesPerImpulse;       (* Rechtsdrehung *)
        IF Dir[CarIndex]<0 THEN Dir[CarIndex]:=Dir[CarIndex]+360; END;
        WITH Pos DO
          Forward(xOld[CarIndex],yOld[CarIndex],x[CarIndex],y[CarIndex],
            MovePerTurn, Dir[CarIndex]); (* Kleine Vorw.Bewegung nach Drehung *)
        END;
      END;
    END;
  END;
END MoveAllCars;

(*-------------------------------------------------------------------------*)

(* Die folgende Prozedur ist für das Plotten aller Fahrzeuges auf dem Grafik-
   bildschirm verantwortlich. Die Bewegung wird als Strichgrafik ausgegeben,
   d.h. für jedes Vehikel wird eine Verbindungslinie zwischen den alten und
   neuen Koordinaten gezogen.                                            *)
PROCEDURE Graphics;

CONST XMax=319; (* Amiga-Grafik: Bildschirmgröße - Breite & Höhe *)
      XMin=0;
      YMax=240;
      YMin=0;

VAR xTest, yTest: BOOLEAN;    (* FÜr Grafik: Testet ob Koord. auf Bildschirm *)
    i: INTEGER;                                        (* Index des Vehikels *)
BEGIN
  FOR i:=1 TO Cars.CarNo DO
    WITH Cars.Coordinate.Pos DO
      xTest:= ((x[i]>XMin) AND (xOld[i]>XMin) AND (x[i]<XMax) AND
               (xOld[i]<XMax));          (* Test: x-Koordinate auf Bildschirm *)
      yTest:= ((y[i]>YMin) AND (yOld[i]>YMin) AND (y[i]<YMax) AND
               (yOld[i]<YMax));          (* Test: y-Koordinate auf Bildschirm *)
      IF (xTest AND yTest) THEN
         (* Wenn Vehikel Nr. i im Bildschirmbereich dann verbinde
            die Punkte [xOld,yOld] und [x,y] in der Farbe "Color".
            Amiga Grafikanweisungen dafÜr: (die y-Verschiebung um 11
            Punkte brauch auf anderen Computern nicht implementiert zu
            werden!)                                                   *)
        SetAPen(drawRP,Cars.Character.Color[i]);
        Move(drawRP,xOld[i],yOld[i]+11);
        Draw(drawRP,x[i],y[i]+11);
      END;
    END;
  END;
END Graphics;

(*************************************************************************)

(* Anfang des Hauptprogramms. Der Grafikbildschirm wird geöffnet und die o.a.
   Prozeduren aufgerufen.                                                *)
```

```
VAR    Digit: DigitalType;                    (* Feld mit Impulsen der Sensoren *)
       ImpulseLeft, ImpulseRight: AllMotorImpulseesType;
                                  (* Feld mit den Impulsen an die Motoren *)
       i: INTEGER;                                    (* Index des Vehikels *)
       CycleIndex, MaxCycles: CARDINAL; (* MaxCycles: Anzahl d. Simulations-
                                                                  schritte *)
       AwaitReturn, Choice: ARRAY [0..0] OF CHAR;          (* siehe Programmtext *)
       Intensity: IntensityType;          (* Feld für die Intensitätsumrechnung *)

BEGIN
  Initialize(Intensity);                (* Initialisierung des Intensitätsfeldes *)
  TermProcedure(Cleanup);  (* Amiga-spezifisch! *)
  DefineCars(); (* Definiere die Vehikel die in der Simulation benutzt werden *)
  Cars.Cycle:=1;                                 (* Setzte Systemtakt auf Anfangswert *)
  LOOP          (* Loop über den ganzen Hauptteil, Abbruch bei Ausführung der
                                                                  EXIT-Anweisung! *)
    Cleanup();       (* Amiga-spezifisch: Evntl. offnen Grafikschirm schließen *)
    WriteLn;
    WriteLn;
    WriteLn;
    WriteString("Optionen: (A)bbruch, (P)ositionseingabe, (W)eiter mit alter ");
    WriteString("Position.");
    WriteLn;
    WriteString("Ihre Wahl --> ");
    ReadString(Choice);
    CASE Choice[0] OF
      "a","A": EXIT; |                                    (* Programmabbruch ! *)
      "p","P": CoordinateInput();                         (* Neue Positionseingabe *)
    ELSE
    END;
    WriteLn;
    WriteString("Wieviele Simulationstakte sollen berechnet werden ?");
    ReadCard(MaxCycles);
    (* Amiga Grafik: Eröffne Grafik-Bildschirm 320x255 *)
      screen:=CreateScreen(320,stdScreenHeight,4,NIL); (* Öffne Bildschirm *)
      Assert(screen#NIL,ADR("Fehler beim öffnen der Grafik !"));
      drawRP:=ADR(screen^.rastPort);    (* Adresse des neuen Grafikbildschirms *)
      LoadRGB4(ADR(screen^.viewPort),ADR(colors),16); (* Setze Farbtabelle *)
    (* Ende Grafikanweisungen *)
    WITH Cars DO
      FOR CycleIndex := 1 TO MaxCycles DO                          (* Taktindex *)
        FOR i := 1 TO CarNo DO                  (* Schleife durchläuft alle Vehikel *)
          IF Character.Brain[i].NeurodNo>=2 THEN (*Kann sich Vehikel bewegen? *)
            IF Character.Sens[i].SensNo#0 THEN                 (* Hat es Sensoren? *)
              SensorCalc(Digit, i, Intensity);   (* Wenn ja, berechne die.. *)
            END;                                          (* ..Impulse von den Sensoren *)
            BrainCalc(Digit, i, ImpulseLeft, ImpulseRight);
                      (* Berechnung der Neurodenverschaltung u. Motorimpulse *)
          END;
        END;
        MoveAllCars(ImpulseLeft,ImpulseRight);
        Graphics();
        IF (Cycle<100) THEN
          INC(Cycle);                   (* Systemtakt läuft zwischen 1 und 100 *)
        ELSE
          Cycle:=1;
        END;
      END;
    END;
    (* Berechnung fertig. Anzeige durch kleines Viereck oben links auf dem
       Grafikbildschirm. *)
    (* Amiga Grafikanweisungen dafür: *)
     SetAPen(drawRP,2); Move(drawRP,2,6);
     Draw(drawRP,2,2); Draw(drawRP,6,2);                    (* Zeichne Viereck *)
     Draw(drawRP,6,6); Draw(drawRP,2,6);
    (* Ende Grafikanweisungen *)
    ReadString(AwaitReturn);                                (* Warte auf RETURN *)
  END;
END Braitenberg.

(*************************************************************************)
```

Bild 6 bis *Bild 9* gedacht. Diese Bilder zeigen komplexe Bahnkurven von Fahrzeugwesen, die sich aus relativ einfachen Neurodenschaltungen ergeben. Dabei stellen *Bild 7* und *Bild 8* Zwischenstadien der Entwicklung eines Fahrzeugwesens dar, dessen Endstadium in Bild 9 in seiner Bewegung dargestellt ist. Allen Fahrzeugwesen (blau in Bild 7, violett in Bild 8 und Bild 9) ist gemeinsam, daß sie wie ein Aggressor, jedoch schneller als dieser, eine Lampe aufsuchen und sich anschließend wieder von ihr lösen können. Im Falle des Vehikels in Bild 9 geschieht dies etwa dadurch, daß ein nach hinten gerichteter dritter Sensor vorhanden ist, der das Überfahren der Lampe registriert.

Die farbigen Abbildungen des Artikels (Bild 1 bis Bild 9) stellen Gesamttrajektorien von Fahrzeugen dar und lassen nur schwer die Dynamik der Fahrzeugbewegung erahnen, die man während des Programmablaufs bei der schrittweisen Ausgabe der Trajektorien beobachten kann. Wer einen schnellen Rechner benutzt, kann eine Änderung am Programm vornehmen, welche die Dynamik noch interessanter macht: man löscht den Bildschirm nach einer bestimmten Anzahl von Taktzyklen, z. B. 5 oder 6. Das ergibt dann den Eindruck von schnellen Würmern auf dem Bildschirm, die sich teilweise überraschend aggressiv verhalten. Dies ist besonders beim Beispiel des Rotators eindrucksvoll.

Wer tiefer in das Programm selbst einsteigen möchte, findet ein Spielzeug mit vielen Möglichkeiten – bis hin zur Definition des eigenen Miniuniversums mit speziellen physikalischen Gesetzen. In dem vorgestellten Programm ist zum Beispiel ein quadratischer Intensitätsabfall der Lichtquellen implementiert. Wenn man hier experimentiert, verändert man die Wahrnehmung und die Braitenberg-Vehikel verhalten sich ganz anders.

Analogien zur Natur

Die Braitenberg-Welt mag auf den ersten Blick realitätsfremd erscheinen. In Braitenbergs Buch [2] finden sich jedoch interessante Parallelen zur Welt der Lebewesen auf der Erde. Betrachtet man z. B. die verschiedenen Bausteine eines Insektenauges, so spiegeln sich einige seiner Eigenschaften in unseren Sensoren wieder. Wie gut bei Insekten so ein zugegebenermaßen komplexer Verband von ca. 20000 Sensoren (Rhabdomere) mit jeweils kleinem Raumwinkelbereich als Auge zu funktionieren vermag, wird jeder bestätigen, der schon einmal vergeblich versuchte, eine Fliege zu fangen.

Da viele Sensoren für sich noch keine Funktion erfüllen, drängt sich die Frage nach der Verarbeitung einer so großen Informationsflut auf. Die Lösung liegt in der hochparallelen Struktur eines biologischen, neuronalen Netzwerkes, die eine Echtzeitsignalverarbeitung auf kleinstem Raum erlaubt. Eine hochparallele Struktur weist auch das Neurodennetz auf, bei dem die eigentliche Verarbeitungsgeschwindigkeit nur von der Tiefe des Neurodenbaumes abhängt.

Desweiteren führt die Frage der Auswertung der Sensorik des Insektenauges direkt zu einer alten Frage der Biologie, warum die Faserbahnen zwischen dem linken Auge und der rechten Hirnhälfte (und umgekehrt) gekreuzt sind. Eine der zur Lösung vorgeschlagenen Theorien, die Theorie von Ramon y Cajal, interpretiert die Kreuzung der Nervenbahnen als notwendige Maßnahme, um die spiegelbildliche Projektion in den Augen wieder zu einem Bild im Gehirn zusammenfügen zu können.

Ohne Kreuzung der Nervenbahnen wandert der linke und der rechte Rand des Urbildes in die Mitte der Projektion, das Urbild wird zerteilt, die Wahrnehmung behindert. Die durch die Kreuzung der Nervenbahnen bewirkte Links/Rechts-Vertauschung der Bilder führt schließlich doch zu einem spiegelbildlich zusammenhängenden Bild.

Professor Braitenberg weist in diesem Zusammenhang darauf hin, daß gekreuzte Nervenbahnen in Wirbeltieren jedoch wahrscheinlich nicht zustande kommen, um, wie im Fliegenauge, die geometrische Optik auszugleichen. Nach seinen Überlegungen sind sie das Resultat einer evolutionären Entwicklung: Die einzigen ungekreuzten Nervenbahnen in den Wirbeltieren führen von der Nase zum Gehirn. Bei primitiven Lebewesen sind geruchsmäßige Sinneseindrücke häufig dominant. Bei unseren frühesten Vorfahren könnten sich dann die gekreuzten Faserbahnen zu den Motoneuronen (Neuronen zur Ansteuerung von Muskeln) ausgebildet haben, da das daraus resultierende Verhalten (siehe die Beispiele dieses Artikels) für das Überleben der Art vorteilhaft war. Das Vorhandensein eines komplexen gekreuzten Nervenstranges könnte dann dazu geführt haben, daß sich die weiteren – in der Entwicklung erst später hinzugekommenen – komplizierteren Verbindungen, z. B. der optischen Sensoren, entsprechend ausgerichtet haben.

Untersuchungen biologischer Informationsverarbeitung in Lebewesen sind wegen der Vielzahl der beteiligten Neuronen sehr schwierig. Computersimulationen können vielleicht eine Möglichkeit bieten, den zugrundeliegenden Vorgängen auf die Spur zu kommen. Die synthetische Psychologie als Folge der Verschaltungen in einfachsten neurodalen Fahrzeugen und deren Computersimulation ist ein erster Schritt auf diesem Weg. Trotz der Primitivität der betrachteten neuro-digitalen Kreaturen ergaben sich bereits bemerkenswerte Resultate. Ist es etwa ein Zufall, daß gerade die Wesen mit gekreuzten Signalleitungen, z. B. der Aggressor, das interessanteste und natürlichste Verhalten zeigen?

Da der Realitätsbezug unserer Simulationen erheblich von der Funktionsweise der Neuroden abhängt, sei schließlich auf deren biologische Grundlage hingewiesen. Eine der ersten Theorien der biologischen Informationsverarbeitung stammt von McCulloch und Pitts [5]. Ihre Theorie beruht auf Beobachtungen der Erregung von Motoneuronen durch Eingangsnerven und formalisiert das Verhalten von Neuronen, deshalb Neuroden genannt. Einige Motoneuronen benötigen Aktivität mehrerer Eingänge, um

```
{ Hier ein Beispiel für die Programmierung eines etwas komplexeren Szenarios:
  das Vehikel Haßliebe zusammen mit einer Lichtquelle                        }

PROCEDURE DefineCars;
BEGIN
  WITH Cars[0] DO
    CarNo:=2;                          (* Anzahl der Vehikel *)
    (* Erstes Vehikel: Lichtquelle *)
    WITH Character DO
      WITH Appear[1] DO
        AppearNo:=1;                                  (* Nur eine Lichtquelle *)
        Appearance[1].AppearIntensity:=32000;         (* Fast max. Intensität *)
        Appearance[1].AppearAngle:=0;                 (* Strahlt in Richtung 0 Grad *)
        Appearance[1].AppearRange:=180;               (* Zwei * 180 Grad = Rundum *)
    END;
    (* Zweites Vehikel: Haßliebender *)
    WITH Sens[2] DO                                   (* Sensoren des zweiten Vehikels *)
      SensNo:=2;                                      (* Zwei Sensoren *)
      Sensor[1].SensorAngle:=70;       (* Linker Sensor 'guckt' nach 70 Grad *)
      Sensor[1].SensorRange:=110;                     (* Sichtbereich +- 110 Grad *)
      Sensor[2].SensorAngle:=-70;    (* Rechter Sensor 'guckt' nach 290 Grad *)
      Sensor[2].SensorRange:=110;                     (* Sichtbereich +- 110 Grad *)
    END;
    WITH Brain[2] DO


      NeurodNo:=9;                                    (* Anzahl der Neuroden *)

      Neurod[1].Threshhold:=1;                        (* Schwellwert des 1. Neurods *)
      Neurod[1].ExcConnectionNo:=1; (* Neurod hat nur eine exz. Verbindung..*)
      Neurod[1].ExcConnections[1]:=-1;                (*.. zum ersten Sensor *)
      Neurod[1].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[2].Threshhold:=1;                        (* Schwellwert des 2. Neurods *)
      Neurod[2].ExcConnectionNo:=1; (* Neurod hat nur eine exz. Verbindung..*)
      Neurod[2].ExcConnections[1]:=-2;                (*.. zum zweiten Sensor *)
      Neurod[2].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[3].Threshhold:=1;                        (* Schwellwert des 3. Neurods *)
      Neurod[3].ExcConnectionNo:=1; (* Neurod hat nur eine exz. Verbindung..*)
      Neurod[3].ExcConnections[1]:=1;                 (* .. zum 1. Neurod *)
      Neurod[3].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[4].Threshhold:=1;                        (* Schwellwert des 4. Neurods *)
      Neurod[4].ExcConnectionNo:=1; (* Neurod hat nur eine exz. Verbindung..*)
      Neurod[4].ExcConnections[1]:=2;                 (* .. zum 2. Neurod *)
      Neurod[4].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[5].Threshhold:=2;                        (* Schwellwert des 5. Neurods *)
      Neurod[5].ExcConnectionNo:=2;  (* Neurod hat zwei exz. Verbindungen.. *)
      Neurod[5].ExcConnections[1]:=1;       (* .. eine zum 1. Neurod und .. *)
      Neurod[5].ExcConnections[2]:=3;       (* .. die andere zum 3. Neurod *)
      Neurod[5].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[6].Threshhold:=2;                        (* Schwellwert des 6. Neurods *)
      Neurod[6].ExcConnectionNo:=2;  (* Neurod hat zwei exz. Verbindungen.. *)
      Neurod[6].ExcConnections[1]:=2;       (* .. eine zum 2. Neurod und .. *)
      Neurod[6].ExcConnections[2]:=4;       (* .. die andere zum 4. Neurod *)
      Neurod[6].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      Neurod[7].Threshhold:=2;                        (* Schwellwert des 7. Neurods *)
      Neurod[7].ExcConnectionNo:=2;  (* Neurod hat zwei exz. Verbindungen.. *)
      Neurod[7].ExcConnections[1]:=5;       (* .. eine zum 5. Neurod und .. *)
      Neurod[7].ExcConnections[2]:=6;       (* .. die andere zum 6. Neurod *)
      Neurod[7].InhConnectionNo:=0;                   (* Keine inh. Verbindungen *)

      (* Vorletztes Neurod gibt Impulse an linken Motor !! *)
      Neurod[8].Threshhold:=1;                        (* Schwellwert des 8. Neurods *)
      Neurod[8].ExcConnectionNo:=2;  (* Neurod hat zwei exz. Verbindungen.. *)
      Neurod[8].ExcConnections[1]:=-2;  (* .. eine zum zweiten Sensor und.. *)
      Neurod[8].ExcConnections[2]:=5;       (* .. die andere zum 5. Neurod *)
      Neurod[8].InhConnectionNo:=2;  (* Neurod hat zwei inh. Verbindungen.. *)
      Neurod[8].InhConnections[1]:=6;       (* .. eine zum 6. Neurod und .. *)
      Neurod[8].InhConnections[2]:=7;       (* .. die andere zum 7. Neurod *)

      (* Letztes Neurod gibt Impulse an rechten Motor !! *)
      Neurod[9].Threshhold:=1;                        (* Schwellwert des 9. Neurods *)
      Neurod[9].ExcConnectionNo:=2;  (* Neurod hat zwei exz. Verbindungen.. *)
      Neurod[9].ExcConnections[1]:=-1;  (* .. eine zum ersten Sensor und.. *)
      Neurod[9].ExcConnections[2]:=6;       (* .. die andere zum 6. Neurod *)
      Neurod[9].InhConnectionNo:=1;(* Neurod hat nur eine inh. Verbindung.. *)
      Neurod[9].InhConnections[1]:=5;       (* .. und zwar zum 5. Neurod *)

    END;
    Color[1]:=1;                          (* Farbcode für die Graphikdarstellung *)
    Color[2]:=2;
  END;
END DefineCars;
```

Bild 19. Beispiel für die Programmierung eines Vehikels: das Haßliebe-Fahrzeug

selbst aktiv zu werden. Dies weist auf die Existenz exzitatorischer Eingänge hin. Andere Motoneuronen benötigen nur einen aktiven Eingang zur Aktivierung. Daraus folgt mit der ersten Beobachtung, daß Neuronen verschiedene Schwellwerte haben können.

Einige Eingänge wirken hemmend auf die Aktivierung des Motoneurons. Neben exzitatorischen Eingängen gibt es also auch inhibitorische Eingänge. Desweiteren beschreiben McCulloch&Pitts das Zusammenwirken der Neuronen mit den Grundbegriffen des Aussagenkalküls: Konjunktion, Disjunktion und Negation, die jeweils durch je ein Neurod darstellbar sind (Bild 15). McCulloch&Pitts' Erkenntnisse sind vollständig ins Neurodenmodell übernommen (siehe auch [1],[2]), so daß man mit dem Programm beschränkt die Signalverarbeitung kleiner neuronaler Netzwerke studieren kann.

Die grundlegende Bedeutung einer solchen Möglichkeit wird erst deutlich, wenn man die von Braitenberg aufgestellte These, das „Gesetz der leichten Synthese und schweren Analyse" [2], bedenkt. Im Rahmen des Braitenbergschen Modells ist leicht einzusehen, daß von der Beobachtung eines Wesens nicht exakt auf seine Struktur geschlossen werden kann. So kann, wie in *Bild 17* gezeigt, ein Fahrzeug mit einem Neurod mit Schwellwert 0, das beide Motoren anregt, nicht von einem Fahrzeug unterschieden werden, in dem jeder Motor von einem eigenen Schwellwert-0-Neurod angetrieben wird. Ebenso arbeiten die zwei in Bild 17 dargestellten Impulsverdoppler trotz verschiedenen inneren Aufbaus nach außen hin gleich.

Ein weiteres Argument für die schwierige Analyse des Zusammenhanges zwischen Verhalten und Neurodennetzen ergibt sich mit den Regeln der Kombinatorik. Nach unten abgeschätzt berechnen sich die Verschaltungsmöglichkeiten eines Wesen mit r Sensoren und n Neuronen, die S zulässige Schwellwerte haben dürfen, zu $(S*V^{(n+r)})^n$. Dabei steht V für die Zahl der Verschaltungstypen von jedem Neurod zu jedem anderen (einschließlich Sensoren und sich selbst). Mit V=3 berücksichtigt man die drei Fälle exzitatorisch, inhibitorisch und nicht verbunden (3 ist zu niedrig, da auch Mehrfachan- und -abregungen sinnvoll sind).

Für das Fahrzeug aus Bild 16 ergeben sich schon rund 10^{50} Verschaltungsmöglichkeiten. Es erscheint also einleuchtend, daß die Analyse eines Wesens aus seinem Verhal-

ten schwieriger ist als die Synthese, bei der man vom Wirken der Baugruppen zu deren Zusammenspiel und zum Test des Wesens gelangt (allerdings ist auch die Synthese nicht mühelos).

Wenn nun schon die Analyse dieser einfachen Wesen derart kompliziert ist, um wieviel schwieriger ist dann die Analyse biologischer Wesen? Braitenberg schlägt hier einen Bogen zur Psychologie, indem er darauf hinweist, daß ein unwissender Beobachter des Braitenbergschen Szenarios hinter den Wesen einen komplizierten Aufbau vermuten würde und bei der Beschreibung der Dynamik des Verhaltens unwillkürlich zu Begriffen aus der Psychologie greift. Es gibt also komplexe Verhaltensweisen, hinter denen sehr einfache „Gehirne" stehen. Braitenberg erhofft sich daraus den Übergang zu einer nüchterneren Begriffswelt in der Psychologie, aus der am Ende ein „besseres Menschenbild" [1] erwachsen soll.

Erweiterungsmöglichkeiten: Geburt, Tod, Evolution

Das oben beschriebene Programm bietet sich zu mannigfaltigen Erweiterungen an, wenn man nicht nur die Spitze des Eisberges aus der Gedankenwelt Braitenbergs erkunden, sondern auch weitere seiner Anregungen computergerecht umsetzen möchte. So ist es beispielsweise relativ einfach, mit ein paar zusätzlichen Regeln zu der Nachbildung von Evolutionsprozessen zu gelangen. Man braucht dazu Geburt und Tod von Wesen, sowie Auswahl- und Mutationsprozesse. Geburt und Tod sind schnell implementiert, weil sie lediglich aus der Kopierfunktion für Wesen mit Erhöhung oder Erniedrigung der Fahrzeuganzahl bestehen.

Ganz anders ist es um das Schaffen der Auswahl- und Mutationsprozesse bestellt. Hier ist Schöpfungskraft und Phantasie erforderlich. Eine einfache Möglichkeit ist, daß das Verlassen des Bildschirmbereiches eines Vehikels seinen Tod und die Geburt eines neuen Wesens bewirkt. Einen anderen Generationsmechanismus erhält man, wenn die Geburt davon abhängig ist, ob sich zwei Fahrzeuge (gleichartige ?) sehr nahe kommen.

Den Tod kann man über das Einführen eines Lebensalterzählers regeln. Schwieriger, aber interessant ist es, den Tod eintreten zu lassen, wenn ein Wesen lange in Bewegungslosigkeit verharrt oder sich auf einer festen Kreisbahn bewegt. Ebenso kann man dafür sorgen, daß das Verschlingen vieler Wesen eine Vervielfältigung der Verschlinger und natürlich den Tod der Verschlungenen zur Folge hat. Einem friedlicheren Wesen mag die Vermehrung auch erlaubt sein, wenn es viele Lampen oder andere Reizquellen aufgesucht hat oder nur lange genug von deren Energie gezehrt hat. Da nicht nur jede einzelne Auswahlregel einführbar ist, sondern auch beliebige Kombinationen möglich sind, ergibt sich schon hieraus eine große Vielfalt.

Auch Mutationsprozesse bei der Geburt von Fahrzeugen lassen dem Schöpfer viel Freiheit, da grundsätzlich alle Fahrzeugeigenschaften mutierbar sind. Sensoren sind im erfaßten Winkelbereich, der Empfindlichkeit, der Anzahl und der Art der Wahrnehmung mutierbar. Exemplarisch sei kurz auf die Wahrscheinlichkeit der einzelnen Mutationsprozesse eingegangen. Die kleine Veränderung ist generell wahrscheinlicher als die große Veränderung, so daß eine leichte Verschiebung des erfaßten Sensorbereiches eher stattfinden sollte als eine große, und die Erhöhung der Sensoranzahl eine seltene Ausnahme bleiben sollte. Niemand sei jedoch gehindert, seinem Spieltrieb folgend, es gerade umgekehrt zu machen.

Das Programm könnte die Veränderung des kleinsten Drehwinkels und der Schrittweite der Motoren ermöglichen. Dazu muß der Datentyp, der jeweils nur einen Wert für alle Fahrzeuge zuläßt, geändert werden. Die meisten Möglichkeiten ergeben sich bei der Mutation des Gehirns. Der Schwellwert der Neuroden, deren Verbindungen und die Anzahl der Neuroden ist veränderlich. Im Erscheinungsbild der Fahrzeuge kann man die Intensität der Reizquelle, deren ausgeleuchteten Winkelbereich oder ihre Anzahl variieren.

Eine leichtere Art der Erweiterung des Programmes bietet das Schaffen neuer Reize und Wahrnehmungsmöglichkeiten. Schöne Beispiele hierzu finden sich in dem Artikel von Dewdney [1]: Über die Einführung einer Farbnummer für Lichtquelle und Sensor kann man den Wesen farbiges Sehen beibringen. In ähnlicher Weise ließen sich Sensoren und Erscheinungsformen für akustische und taktile Reize sowie Rezeptoren zur Wahrnehmung von Wärme oder Energieverteilung im Raum implementieren. Reizübertragungen könnten durch Umwelteinflüsse wie Wind (für Geräusche), Diffusion usw. verändert werden.

Der letzte Vorschlag für eine Erweiterung des Programmes läuft auf eine stärkere Anpassung der Neuroden an wirkliche Neuronen hinaus. Neuronen feuern nicht wie Neuroden in unserem Modell zu einem bestimmten Zeitpunkt. Der Systemtakt, die angenommene innere Uhr, existiert für wirkliche Neuronen nicht. Daher ist es nötig, eine Signallaufzeit von Neurod zu Neurod festzulegen. In neuronalen Netzwerken ist diese jedoch keineswegs konstant."

An dieser Stelle endet die gespeicherte Programmdokumentation des Enterprise-Bordcomputers. Spock ist fasziniert von der Möglichkeit, mit diesem Programm die weitere Entwicklung auf dem Planeten vorherzusehen, und macht sich mit großem Eifer daran, einige Simulationen auszuarbeiten. Captain Kirk ist überzeugt, daß das vorgefundene Programm ausreichen wird, um die Bewohner dieser seltsamen Welt besser zu verstehen. Er gibt daher Befehl, den Orbit zu verlassen, und wenig später ist die Enterprise wieder unterwegs, abermals auf der Suche nach neuen Welten und erstaunlichen Lebensformen.

Viel Spaß nun all denen, die sich in die Wunderwelt der synthetischen Psychologie wagen wollen.

Literatur

[1] *Dewdney, A.K.:* Computer Recreations – Braitenberg memoirs: vehicles for probing behavior roam a dark plain marked by lights. Scientific American, März 1987.

[2] *Braitenberg, Valentin:* Künstliche Wesen – Verhalten kybernetischer Vehikel. Vieweg Verlag, Wiesbaden 1986.

[3] *Ein sehr gutes Modula-2 System für Amiga:* M2-Amiga Software Development System von A.+L. Meier-Vogt.

[4] Das ungekürzte Programm kann auch auf Diskette bezogen werden: Um den Aufwand in Grenzen zu halten, bitten wir um Zusendung einer leeren Diskette (Amiga 3½" DS/DD bzw. IBM 5¼" 360 KB) mit frankiertem und selbstadressiertem Rückumschlag sowie einer Schutzgebühr von 10 DM (Schein) an folgende Adresse:
Christoph Köhler
Zeisigstr. 30c
8011 Vaterstetten
(Bitte außen auf dem Umschlag Computertyp angeben.)

[5] *McCulloch, W.S. & Pitts, W.H.:* A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, p 115, 1943.