Der Aufbau einer komplexen Umwelt für den Multi-Agenten-Betrieb und Hybridgesellschaften

Jürgen Gerdes & Frank Detje

Institut für Theoretische Psychologie, Universität Bamberg

[juergen.gerdes | frank.detje]@ppp.uni-bamberg.de

1. Sozionische Forschungsfragen und komplexe Umwelten

Unser sozionisches Forschungsprojekt betrifft in der Hauptsache die Frage nach der Entstehung und Änderung sozialer Strukturen in Gruppen intelligenter, multimotivierter, emotionaler Agenten. Damit ist gemeint, dass wir den bisherigen Rahmen indidualpsychologischer Untersuchungen mit der Implementation einer Theorie (PSI) menschlicher Absichts- bzw. Handlungsregulation (Dörner, 1999) um soziologische Untersuchungen erweitern möchten. Das PSI-Programm (Dörner & Gerdes, 1998-2001) wird hierfür von seiner bisherigen Umwelt („Insel“, siehe Detje, 2000a) getrennt und zum Multi-Agenten-Betrieb befähigt. Das PSI-Programm ist bereits in der Lage, in seiner Umwelt autonom zu handeln und wir konnten bereits nachweisen, dass sich dieser PSI-Agent („James“) ähnlich verhält wie Menschen in derselben Umgebung (Detje, 2000b; Dörner, 2000). Parallel zum Ausbau von PSI wird eine neue Umwelt für die Implementation der Theorie geschaffen, die es erlaubt, mehrere autonome Agenten in ihr handeln zu lassen.

Doch unser Forschungsvorhaben betrifft nicht alleine die Frage nach Prozessen der Gesellschaftsbildung und des –zerfalls bei künstlichen autonomen Agenten. Tatsächlich ist unser Anliegen ein Dreigleisiges:

1) Wir möchten eine Umwelt schaffen und das PSI-Programm erweitern, um eine homogene Menge künstlicher autonomer Agenten unter verschiedenen Bedingungen zu beobachten, um Determinanten für Gruppenbildung, Gruppenstabilität, Gruppenstrukturen, Gruppenzerfall und dergleichen analysieren zu können. Die Bedingungsvariation betrifft dabei sowohl die Eigenschaften der Umwelt, die Fähigkeitsprofile der einzelnen Agenten als auch die Persönlichkeitseigenschaften dieser Agenten.

2) Wir möchten eine Simulationsumgebung schaffen, die es erlaubt, verschiedene menschliche Versuchspersonen in einem Multi-Agenten-Betrieb in ihr agieren zu lassen. Dies ist notwendig, um festzustellen, welche Gruppenprozesse bei Menschen tatsächlich in einer solchen Umwelt zu beobachten sind. Vor allem interessiert uns dabei, a) einzelne Phänomene zu beobachten und zu analysieren, um festuzstellen, ob die im PSI-Programm integrierten Konzepte ausreichen, diese Phänomene zu erklären und abzubilden, oder ob die dahinterstehende Theorie bzw. ihre Implementation noch „weisse Flecken“ aufweist; b) die Verhaltensvariabilität der einzelnen Menschen und der verschiedenen Gruppen, die sie bilden, zu beobachten und zu analysieren, um festzustellen, ob die Theorie und ihre Implementation in der Lage ist, die beobachtete Spannbreite an Verhaltensweisen ebenfalls abzubilden und zu erklären, oder ob sie trotz Variation der Parameter ein zu einheitliches Verhalten zeigt.

3) Wir möchten eine Umgebung schaffen, in der es möglich ist, heterogene Gruppen aus autonomen PSI-Agenten und Menschen miteinander leben zu lassen. Wichtigstes Forschungsziel ist es, die PSI-Theorie und das PSI-Programm um bisher fehlende Konzepte so zu erweitern, dass im Rahmen des sozionischen Forschungsprojekts auch das Verhalten unseres künstlichen Systems in Bezug auf Gruppenprozesse menschliche Züge annimmt. Gerade diese Form der Hybridgesellschaften erlauben uns eine diesbezügliche Analyse. In einem Turing-ähnlichen Szenario (Turing, 1950) werden wir die beteiligten menschlichen Personen danach befragen, welche der anderen Agenten in der Umgebung menschliche bzw. künstliche Agenten sind. Sollte es keine überzufällig richtige Zuordnung geben, ist dies Zeichen dafür, dass sich beide Agentenpopulationen, trotz aller Varianz im Verhalten, hinreichend ähnlich sind. Sollte die Zuordnung überzufällig richtig sein, so können wir danach fragen, welche der Verhaltensweisen, Agenteneigenschaften, etc. noch zu wenig menschlich sind, um wiederum Aufschluss für Veränderungen der Theorie und ihrer Implementation zu bekommen.

Diesen drei Zielen möchten wir mit nur einem einzigen Umweltszenario nachgehen. Hierfür muss dieses Umweltszenario bestimmte Eigenschaften besitzen, die adäquate sozionische Untersuchungen mit homogenen Gruppen menschlicher und künstlicher Agenten aber auch mit Hybridgesellschaften erlauben. Deshalb muss das Szenario auf der einen Seite „komplex“ sein (siehe z.B. Dörner, 1989), auf der anderen Seite muss es aber einfach zu bedienen, zu erlernen und wahrzunehmen sein. Hinzu kommen die Notwendigkeit einer möglichst einfachen Veränderung sämtlicher Eigenschaften des Umweltszenarios zur Variation der Bedingungen und eine vollständige Protokollierung sämtlicher Eingriffe aller in ihr handelnden Agenten zur Analyse individuellen Verhaltens und der unterschiedlichen Gruppenprozesse.

Die geforderte „Komplexität“ der Umwelt betrifft ganz unterschiedliche Merkmale des Szenarios. Die Umwelt muss

- in ihrer *räumlichen Ausdehnung* gross genug sein, so dass sie nicht in kurzer Zeit bereits vollständig exploriert ist;
- *intransparente* Eigenschaften aufweisen, d.h. nicht alle implementierten Wechselwirkungen dürfen sofort durchschaubar sein, damit es „etwas zu entdecken“ gibt;
- *viele unterschiedliche Verhaltensmöglichkeiten* erfordern und möglich machen, damit die potentielle Variabilität individuellen Verhaltens möglichst groß wird;
- *unterschiedliche Aufgaben und Probleme* bereit halten, damit verschiedene Aspekte von Gruppenprozessen unter kontrollierten Bedingungen erhoben werden können;
- *von Struktur und Aufbau heterogen* geschaffen sein, damit sie unterschiedliche Herausforderungen an Prozesse des Planens, der Erwartungsbildung, der Konfliktbewältigung, usw. bereit hält;
- *eigendynamische* Komponenten bereitstellen, damit a) bei kurzfristig wirkenden Dynamiken „Schreck“ und „Überraschung“ möglich werden und b) bei langfristigen „Totzeiten“ für die Agenten Planungen erforderlich werden;

- *eine große Vielzahl sowohl unterschiedlicher als auch ähnlicher Objekte* beinhalten, damit unterschiedliche kognitive Prozesse der Gedächtnisbildung sowie Verwechslungen der Umweltgegenstände analysiert werden können.

Zusammenfassend und umgangssprachlich ausgedrückt, wird also gefordert, dass die Umwelt für die menschlichen Agenten genügend Herausforderung und Spannung bietet, von den künstlichen Agenten bewältigt werden kann, und für die Beantwortung sozionischer Fragestellungen geschaffen ist.

Wir haben uns entschlossen, für unsere Untersuchungsziele das Konzept eines Abenteuerspiel-ähnlichen Szenarios in Form einer „Insel“ beizubehalten, das bestehende Konzept einer „Insel“ aber weitgehend umzugestalten. Die massgeblichen Veränderungen betreffen jedoch nicht nur das Potenzial zum Multi-Agenten-Betrieb, sondern weitere Veränderungen, die die Verhaltensvielfalt, Größe der Umwelt, Eigendynamik und dergleichen betreffen. Das auf die oben genannten Kriterien hin entwickelte Szenario „Psi 3D“ (Gerdes & Dörner, 2001) wird in den folgenden Abschnitten näher erläutert.

2. Datenstrukturen einer entsprechenden Umwelt

„Psi 3D“ ist ein Programm zur Konstruktion und Simulation von verschiedenen Umwelten für menschliche Versuchspersonen, künstliche Agenten oder hybride Gruppen. Eine Lösung der Aufgabe, eine nicht triviale Umwelt zu schaffen, die von ganz unterschiedlich zusammen gesetzten Gruppen bewältigt werden kann, wurde in diesem Projekt auf eine allgemeine Grundlage gestellt.

Das realisierte Programm „Psi 3D“ gibt uns die Möglichkeit, dieselbe Umwelt sowohl für Versuchspersonen, als auch für künstliche Agenten zu benutzen. Somit lässt sich das Verhalten von Versuchspersonen und künstlichen Agenten unmittelbar vergleichen. Ebenso bietet dieses Programm die Möglichkeit von mehreren Rechnern aus menschliche und künstliche Agenten gleichzeitig in derselben Umwelt agieren zu lassen, so dass auch hybride Gruppen diese Umwelt für turing-ähnliche Tests bearbeiten können.

Der Multi-Agenten-Betrieb wird über eine Server-Client-Konzeption ermöglicht. Jeder auf den Server zugreifenden Client besitzt ein eigenes Abbild der Umwelt, in der er und die anderen Clients (Agenten) integriert sind. Für jeden Agenten sind alle anderen Agenten Bestandteil seiner Umwelt. Die Idee, die Umwelt nur einmal, zentral im Server zu verwalten, wurde verworfen, da dies mit einer sehr hohe Kommunikationsfrequenz der einzelnen Clients mit dem Sever einhergegangen wäre. Jede Bewegung im Raum der Umwelt geht z.B. mit einer unterschiedlichen (dreidimansionalen) Ansicht aller im Blickfeld befindlichen Objekte einher. Der scheinbar umständlichere Fall, dass alle Clients ihr eigenes Abbild der Umwelt besitzen, führt dazu, dass viele Berechnungen nun lokal durchgeführt werden können (dies betrifft vor allem die graphische Darstellungen der Umwelt), was zu einer wesentlich erhöhten zeitlichen Effizienz des Gesamtbetriebs führt.

Jeder Client kann zu jedem Zeitpunkt unabhängig von den anderen Clients in der Umwelt agieren. Die Clients senden ihre jeweils als Handlungen zu interpretiernden Aktionen (z.B.

eine Drehung um 10°, das Aufheben von Objekten, etc) asynchron über TCP/IP an den Server. Der Server wiederum gibt diese Informationen, im Fall, dass diese Änderung tatsächlich durchgeführt werden kann, dann an alle Clients weiter. Der Server besitzt hierfür ein eigenes Abbild der Umwelt, und zwar das jeweils aktuellste. Die Clients nehmen die angeforderten Änderungen synchron (Broadcasting) vom Server entgegen und modifizieren ihre Abbilder der Umwelt entsprechend.

Sollten zwei Clients aber z.B. denselben Ort innerhalb der Umgebung relativ gleichzeitig einnehmen wollen (asynchrone Meldung eines „Move"-Ereignisses noch bevor die Abbilder aktualisiert werden konnten), so wird es trotzdem keinen „Konflikt" geben, da vom Server immer nur ein Ereignis zur Zeit verarbeitet wird. Der erste „Move"-Befehl führt zu einer Bewegung, die zweite „Move"-Meldung aber wird als „undurchführbarer Änderungswunsch" dem Client2 gemeldet. Client1 kommt also an den gewünschten Ort; Client2 hingegen kollidert mit Client1 und bewegt sich nicht (sondern nimmt gegebenenfalls sogar Schaden).

Als nächstes soll das Umweltszenario näher beschrieben werden, da es die sozionischen Anforderungen an unser Projekt realsiert.

Das Umweltmodul „Psi 3D" enthält selbst keine Angaben über eine spezifische Umwelt. Es bildet viel mehr einen mathematischen Kern ab, der es erlaubt, eine Vielzahl verschiedener Umwelten zu verarbeiten. Damit können für unterschiedliche Fragestellungen jeweils spezifische Umwelten geschaffen werden, ohne dass am Programmcode etwas geändert werden müsste. Es sind die einzulesenden Konfigurationsdateien, die hierfür lediglich angepasst werden müssen. „Psi 3D" liest diese Dateien beim Programmstart ein, erlaubt aber über eine eigene Eingabeoberfläche die online-Veränderung sämtlicher Eigenschaften, das betrifft auch das Hinzufügen neuer Objekte, Objekttypen, und das Löschen von Objekten.

Die Protokollierung des Verhaltens aller Agenten (künstlich oder menschlich) und der eigendynamischen Umweltänderungen ist vollständig, so dass auf Individuen und auf Gruppen bezogene Fragen mit diesen Daten analysiert werden können.

Die Umwelt[1] besteht insgesamt aus nur drei verschiedenen Klassen von Konzepten:

a) es gibt eine Grundfläche, die unter anderem die räumliche Ausdehnung der Umwelt repräsentiert,

b) es gibt auf dieser Grundfläche Objekte, die ganz verschiedener Art sein können,

c) es gibt Möglichkeiten zur Veränderung der Umwelt, sei es durch Eingriffe einzelner Agenten oder eigendynamischer Veränderungen der Umwelt, die über „Nachrichten" (siehe unten) ausgelöst werden, also durch „Kommunikation" der Objekte mit ihrer Umwelt zustande kommen.

Diese drei Konzepte sind jeweils sehr einfach realisiert, d.h. allgemein aufgebaut und dadurch flexibel angelegt, so dass auf relativ einfache Weise Szenarien mit völlig unterschiedlichen Eigenschaften konstruiert werden können. Der Reihe nach werden wir nun auf die techni-

[1] Die aktuelle Umwelt wurde von Viola Hämmer erstellt. Diese Umwelt enthält zur Zeit ca. 1800 Dreieckspunkte, 3500 Dreiecke, 1000 Objekte, 200 Objekttypen und 330 Aktionen (siehe Erläuterungen weiter unten).

schen Aspekte dieser drei Grundkonzepte näher eingehen. Alle zur Realisierung einer bestimmten Umwelt notwendigen Datendateien des „Psi 3D“-Programms befinden sich im Unterverzeichnis 'Daten'.

3. Die Grundfläche

3.1 Struktur der Grundfläche

Die Grundfläche der Umwelt besteht aus verschiedenen Dreiecken. Dreiecke wiederum werden durch Dreieckspunkte mit jeweils 3 Koordinaten und einer Farbe festgelegt. Abbildung 1 zeigt beispielhaft die Draufsicht einer konkreten Realisierung einer solchen Umwelt.

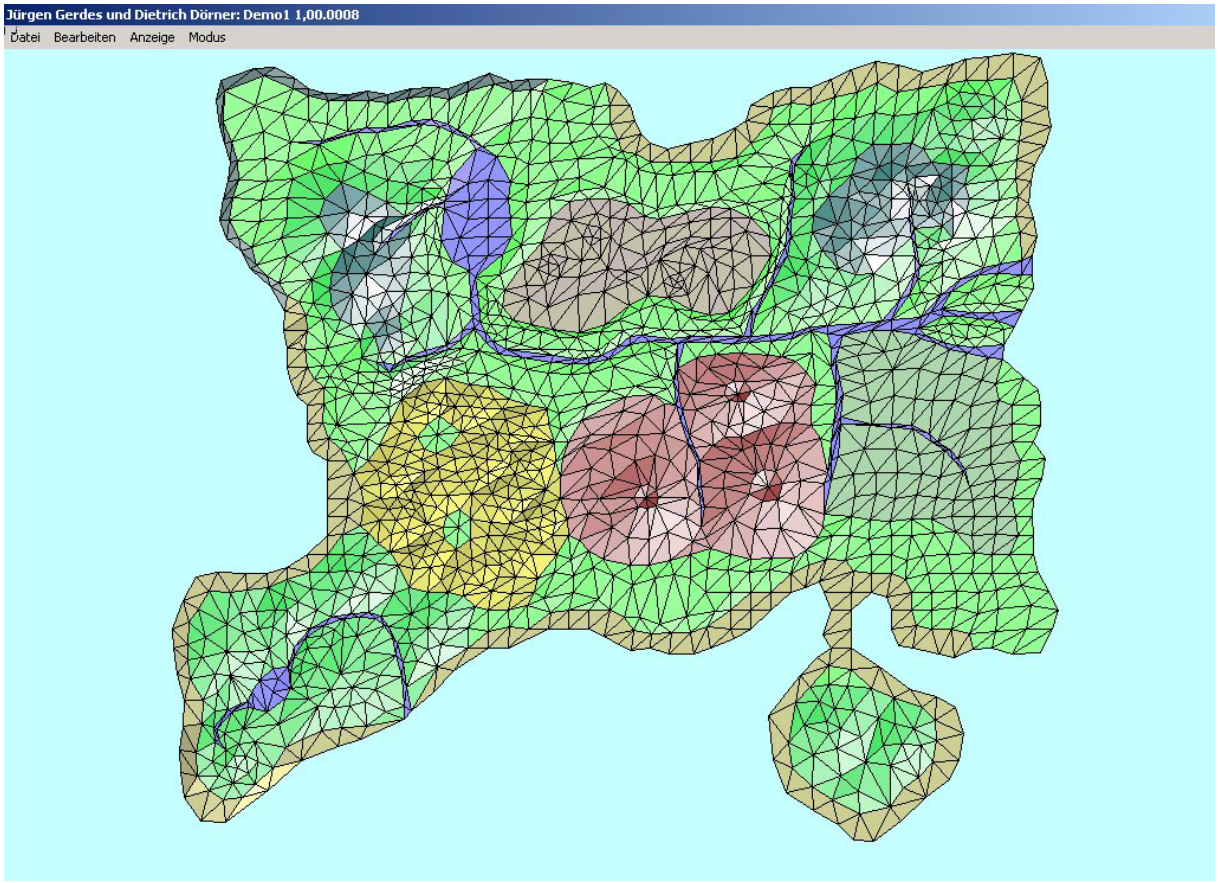

Abbildung 1: Die Grundfläche der Umwelt, bestehend aus farbigen Dreiecken.

Die Dreiecke sind jeweils einem bestimmten Objekttyp zugeordnet. Objekttypen haben jeweils einen bestimmten, beliebigen, vom Benutzer definierten, Namen. Dieser Objekttyp legt die spezifischen Eigenschaften eines Dreiecks fest. Für Dreiecke ist dies unter anderem die Farbe.

Die gelblichen Dreiecke links von der Inselmitte gehören z.B. dem Objekttyp 'Wüste' an, jene rechts daneben gehören zum Objekttyp 'Vulkan'. Eine weitere, für Dreiecks-Objekttypen relevante Eigenschaft ist der Geschwindigkeitsfaktor, der die Geschwindigkeit beeinflusst, mit der sich ein bewegliches Objekt über diese Oberfläche bewegt (z.B. kommt man trotz gleichem Energieaufwand über Sandflächen schwerer, d.h. langsamer, voran als über Wege).

3.2 Definition der Grundfläche

In der Datei *'TrianglePoints.Dat'* sind alle Dreieckspunkte aufgeführt. In jeder Zeile stehen die Koordinaten x, y, z eines Punktes. Die letzte Zahl in einer Zeile gibt lediglich die Indexnummer des Punktes an (siehe Definition 1). Sie wird beim Einlesen nicht berücksichtigt.

```
1500.000 100.000 0.000 15
```

Defnition 1: Dreieckspunkte (Datei „TrianglePoints.Dat“).

Die Dreiecke werden in *'Triangles.Dat'* definiert. In jeder Zeile stehen die Indizes der Dreieckspunkte, aus denen ein spezifisches Dreieck gebildet werden soll, dessen Objekttyp und die Indexnummer des Dreiecks selbst (siehe Definition 2).

```
1277 1276 1279 Wüste 1951
```

Defnition 2: Dreiecke (Datei „Triangles.Dat“).

Alle weiteren Eigenschaften, die Dreiecken zugeordnet werden können, sind im Objekttyp festgelegt, z.B. deren oben angesprochene Farbe, mit der sie als Bestandteil der Grundfläche (des Untergrundes) dargestellt werden. Es sind also nicht bestimmte Dreiecke „gelb“, sondern z.B. die Dreiecke des Objekttyps „Wüste“ (siehe 4.2 Definition der Objekte).

Dadurch, dass Dreieckspunkte getrennt von den „eigentlichen“ Dreiecken indiziert und angegeben werden, erhalten wir eine Konfiguration (eine Fläche) zusammenhängender Dreiecke. Verschiedene Dreiecke dürfen auf gleiche Dreieckspunkte verweisen. Nicht durch Dreiecke verwendete Dreieckspunkte, werden ignoriert. Fehler bei der Definition von Dreiecken, die dadurch entstehen, dass bestimmte Punktkombinationen zu sich überschneidenden Dreiecken führen, werden vom Programm automatisch angezeigt (und deren Behebung empfohlen). Dreiecke werden behandelt wie Objekte (siehe unten) auch. Dreiecke können zu Ebenen mit unterschiedlichen Höhen und Tiefen geformt werden; „überdachte“ Formationen zu definieren, wie z.B. Höhlen, ist mit dieser Konzeption jedoch nicht möglich.

4. Die Objekte

4.1 Struktur der Objekte

Die verschiedenen Objekte, die es in der Umwelt geben soll, stehen jeweils auf einem der vorher definierten Dreiecke und hier an einem beliebigen Ort auf dem Dreieck. Deshalb können auch mehrere Objekte auf einem Dreieck stehen. Es werden für die Objekte nur die x- und y-Koordinate angegeben, die z-Koordinate wird aufgrund der Dreiecksdefinition vom Programm errechnet. Abbildung 2 zeigt beispielhaft (in einer dreidimensionalen Konfiguration) verschiedene Objekte, die auf den Dreiecken der Grundfläche der Abbildung 1 stehen.

Auch die Objekte sind jeweils einem bestimmten Objekttyp zugeordnet, der die spezifischen Eigenschaften der Objekte festlegt. Dazu gehören der Name des Objekttyps, seine verschiedenen Ansichten bei der Darstellung in der dreidimensionalen Umwelt, bestimmte Variablenwerte, auf die wir gleich näher eingehen, und die Nachrichten, die von diesem Objekttyp verarbeitet werden können, auf die wir weiter unten eingehen werden.

Abbildung 2: Dreidimensionale Ansicht der Umwelt mit Objekten.

Die Objekte links im Bild werden z.B. durch die Objekttypen mit den Namen 'Palme 1' und 'Palme 2' definiert. 'Palme 1' ist bereits abgeerntet, während auf 'Palme 2' noch Nüsse zu finden sind. Die Eigenschaften, die sich hier unterscheiden ist der Bitmap-Name des darzustellendes Objektes. Hieraus erwachsen aber noch Konsequenzen: so ist es z.B. nicht möglich „Palme 1“ zu ernten, was für „Palme 2“ möglich ist; „Palme 2“ kann nach dem Ernte-Vorgang den Objekttyp „Palme 1“ annehmen, mit all dessen Eigenschaften.

Die dreidimensionale Ansicht der Objekte wird durch eine oder mehrere Bitmaps definiert. So lassen sich relativ einfach Ansichten aus verschiedener Perspektive realisieren. Abbildung 3 zeigt beispielhaft vier verschiedene Ansichten des Objekts „Löschfahrzeug01“. Bei der Erstellung einer Ansicht wird die Bewegungsrichtung des Objektes und die Blickrichtung des Beobachters berücksichtigt. Die jeweils passenste Bitmap wird ausgewählt und dargestellt. Die Anzahl der Ansichten ist beliebig.

Abbildung 3: Verschiedene Ansichten eines Objekts.

Jedes Objekt in der Umwelt hat auch die Eigenschaft, ausgedehnt zu sein, wobei durch die Angabe eines Radius die Ausdehnung des Objekts bestimmt wird. Damit ist z.B. sichergestellt, dass Objekte miteinander kollidieren können, bzw. sich gegebenenfalls gegenseitig ausweichen müssen.

Objekte können sich in der Umwelt bewegen oder an einem bestimmten Platz verbleiben. Für bewegliche Objekte sind die Variablen Schrittweite und Steigung relevant. Die Schrittweite gibt an, wie weit sich ein Objekt bei einem Schritt bewegt, also wie schnell es sein kann. Die Steigung gibt an, welche Steigungen oder Gefälle gerade noch überwunden werden können. Sind Steigung oder Gefälle größer als dieser Wert, kann sich das Objekt nicht in die gewünschte (angeforderte) Richtung bewegen.

Der Geschwindigkeitsfaktor gibt an, mit welchem Faktor die Schrittweite eines Objektes multipliziert wird, das sich auf diesem Objekt bewegt. Dem Objekttyp ‚Wüste', einem gelben Dreieck, sei der Geschwindigkeitsfaktor 0.5 zugeordnet. Bewegt sich ‚Löschfahrzeug01' normalerweise mit einer Schrittweite von 10.0 Einheiten, so schafft es auf der ‚Wüste' nur 5.0 Einheiten. Ist der Wert dieser Variablen 0.0, kann das Objekt definitionsgemäß gar nicht erst betreten werden („Löschfahrzeug01" nimmt hier z.B. den Wert „0.0" an).

Objekte können aber auch andere Objekte aufnehmen und diese wieder in die Umwelt zurück legen. Jedes Objekt besitzt Angaben über das Gewicht und das Volumen. Die Variablen MaxGewicht und MaxVolumen bestimmen die Aufnahmekapazität eines Objekts. Auf diese Weise haben wir das Konzept des „Gepäcks" realisiert, dass für die Agenten z.B. wichtig wird, wenn sie grosse Wüstenflächen überqueren wollen, in denen es kaum Wasser gibt.

4.2 Definition der Objekte

Objekte werden in der Datei Objects.Dat' definiert. In jeder Zeile stehen die Koordinaten x, y, z, der horizontale, der vertikale Richtungswinkel und die Sichtbarkeit eines Objekts, der Objekttyp und die Indexnummer (siehe Definition 3).

```
3319.27 -634.17 2.06 0.00 0.00 1 Sonnenblume 1047
```

Definition 3: Objekte (Datei „Objects.Dat").

Objekttypen werden in der Datei 'ObjectTypes.Dat' definiert. Die jeweiligen Definitionen der verschiedenen Objekttypen ist in mehrere Abschnitte unterteilt. Der erste Abschnitt beschreibt den Namen des Objekttyps (siehe Definition 4). Über diesen „Namen" erfolgt die Zuordnung von Objekten zu ihren Eigenschaften, die der jeweilige Objekttyp beschreibt.

```
NAME
Hügel
```

Defnition 4: Objekttypen I (Datei „ObjectTypes.Dat").

Im nächsten Abschnitt folgen Angaben über die verschiedene Ansichten (siehe Definition 5). Diese Angaben entsprechen Namen von Bitmap-Dateien, die im Unterverzeichnis 'Bitmaps' vorhanden sein müssen. Diese Ansichten sind nur für Objekte, nicht für Dreiecke, relevant.

```
ANSICHT
Psi-nachhinten
Psi-nachrechts
Psi-nachvorne
Psi-nachlinks
```

Defnition 5: Objekttypen II (Datei „ObjectTypes.Dat“).

Es folgt der Abschnitt, der die Werte für die Variablen der Objekteigenschaften spezifiziert (siehe Definition 6). Auf die Bedeutung dieser Variablen sind wir weiter oben schon eingegangen.

```
VARIABLEN
   50.000 Radius
1.000 Geschwindigkeitsfaktor
  100.000 Gewicht
  100.000 Volumen
  $002BDC05 Farbe
0.000 Schrittweite
  0.000 Steigung
 0.000 MaxGewicht
 0.000 MaxVolumen
```

Defnition 6: Objekttypen III (Datei „ObjectTypes.Dat“).

Objekte können über Nachrichten miteinander und mit der Umwelt „kommunizieren“. Diese Kommunikation betrifft z.B. die Bewegung durch die Insel-Landschaft (z.B. durch eine „Move“-Nachricht), die Manipulation von Objekten (z.B. durch eine „Hämmern“-Nachricht) oder Agenten (z.B. durch eine „Küssen“-Nachricht). Die Nachrichten führen beim empfangenden Objekt zu bestimmten Aktionen. Der Empfänger wird spezifiziert durch die Angabe des Objekttyps, so dass ein sendendes Objekt auch gleichzeitig Empfänger der Nachricht sein kann, was z.B. bei einer „Anünden“-Nachricht dazu führen kann, dass bei einem Waldbrand sowohl andere Bäume anfangen, Feuer zu fangen, das sendende Objekt aber weiter abbrennt. Das Konzept dieser Nachrichten wird im folgenden Abschnitt näher beschrieben.

5. Das Nachrichtenkonzept

Jedes Objekt wird – wie gesagt – durch einen bestimmten Objekttyp beschrieben. Ein Objekt kann bestimmte Nachrichten senden und auf bestimmte Nachrichten reagieren.

```
Objekttyp
   Ziel
   Name
Wahrscheinlichkeit
   Dauer
  MsgSelf
 MsgOther
 AutoStart
AutoStartRandom
 AutoRepeat
```

Defnition 7: Nachrichten (Datei „Messages.Dat“).

Die für ein bestimmtes Objekt zulässigen Nachrichten werden durch seinen Objekttyp definiert. Definition 7 zeigt die Datenstruktur des Nachrichtenkonzepts, Definition 8 ein konkretes Beispiel für eine Nachricht, die dem Objekttyp „Akazie“ zugeordnet wird.

```
Akazie
Akaziebrennt1
Anzünden
0.50000000
4 s
Akaziebrennt1
Anzünden 150
0
```

Defnition 8: Konkretes Beispiel für eine definierte Nachricht (Datei „Messages.Dat“).

Die Datei „Messages.Dat“ enthält die Defnitionen aller festgelegten Nachrichten. Diese Nachrichten werden den oben besprochenen Objekttypen durch die Angabe des Namens des Objekttyps zugeordnet (1. Zeile in Definition 7 und 8).

Eine ‚Akazie’ reagiert auf die Nachricht mit dem Namen ‚Anzünden’ (Zeile 3) wie folgt: sie ändert sich nach 4 s (Zeile 5) mit einer Wahrscheinlichkeit von p=0.5 (Zeile 4) in den Objekttyp ‚AkazieBrennt1’ (Zeile 2). Dabei sendet sie die Nachricht, mit dem (gleichlautenden Namen „Akaziebrennt1“) an sich selber (Zeile 6). Das bedeutet in diesem Fall, aber in Definition 8 nicht angegeben: „Akaziebrennt1“ verwandelt sich wiederum nach einer bestimmten Zeit und mit einer gewissen Wahrscheinlichkeit um in den Objekttyp „Akaziebrennt2“ (definiert in der Nachrichtendefinition für den Objekttyp „AkazieBrennt1“), d.h. die Akazie brennt weiter ab. Gleichzeitig sendet „AkazieBrennt1“ an alle anderen Objekte im Abstand von 150 die Nachricht ‚Anzünden’ (Zeile 7). Die „0“ in Zeile 8 Definition 8 fasst die letzten drei Zeilen von Definition 7 zusammen. Dies sind in Definition 8 die booleschen Variablen AutoStart, AutoStartRandom und AutoRepeat. Eine „0“ bedeutet hier, dass alle „Flags“ auf „aus“ geschaltet werden, eine „7“ bedeutet, dass alle auf „an“ geschaltet werden. Interpretiert man die Reihenfolge der drei Flags als eine Anordnung aus drei Bits, so lässt sich eine eineindeutige Festlegung aller drei Flag-Werte durch die Angabe einer einzigen Integerzahl erreichen.

Da auch das Aussehen (d.h. die in „Ansichten“ definierten Bitmap-Dateinamen) eines Objektes im Objekttyp angegeben wird, ist es möglich, identisch aussehende Objekte auf unterschiedliche Nachrichten reagieren zu lassen. Konzeptuell gesehen, erhöhen wir auf diese Weise die Flexibilität des Szenarios, den Spielablauf betreffend können wir auf diese Weise unseren Agenten identisch aussehende, aber nicht identisch zu behandelnde Objekte vorgeben. D.h., einmal als zielführend gelernte Verhaltensweisen sind „plötzlich“ nicht mehr adäquat. Ebenso können verschiedene Objekte auf die gleiche Nachricht unterschiedlich reagieren.

Objekte „kommunizieren“ ausschliesslich über Nachrichten miteinander. Kann ein Objekt eine Nachricht verarbeiten (d.h. die Nachricht ist in seinem Objekttyp definiert), werden bestimmte Aktionen ausgeführt (siehe Definition 7 und 8).

Im *einfachsten* Fall führt der Empfang einer Nachricht sofort zu einer Änderung des Objekttyps des empfangenden Objektes.

Nehmen wir an, einem Objekt des Typs 'Palme' wird die Nachricht 'fällen' gesandt. Daraufhin wandelt sie sich in ein anderes Objekt 'Palme gefällt' um (siehe Abbildung 4). Der neue Objekttyp wird in der Variablen „Ziel“ definiert.

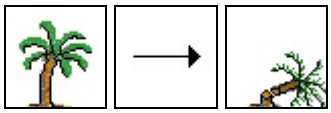

Abbildung 4: Ein Objekt verwandelt sich.

Mit der Variablen „Dauer“ lässt sich die für diese Aktion benötigte Zeit einstellen (siehe Abbildung 5), d.h. die Änderung tritt nach dieser Zeit erst sichtbar ein.

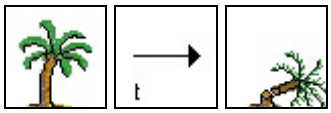

Abbildung 5: Ein Objekt verwandelt sich nach gewisser Zeit.

Für die Zeitangaben ist die Verwendung verschiedener Einheiten erlaubt (siehe Definition 9).

```
Sekunden: s, sec oder sek
  Minuten:  m oder min
 Stunden:  h oder hour
  Tage:     d oder day
 Monate:   M oder month
 Jahre:    y oder year
```

Defnition 9: Erlaubte Zeitangaben (Datei „ObjectTypes.Dat“).

Nicht jede Aktion (Senden einer Nachricht) muss zum Erfolg (d.h. zu einer Änderung des Objektes) führen (siehe Abbildung 6). Dies wird über die „Wahrscheinlichkeit“ geregelt. Deren Wert liegt zwischen 0.0 und 1.0.

Abbildung 6: Ein Objekt verwandelt sich nach gewisser Zeit mit einer gewissen Wahrscheinlichkeit.

Eine bereits gefällte Palme reagiert ebenfalls nicht auf die Nachricht 'fällen' (siehe Abbildung 7).

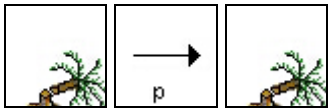

Abbildung 7: Ein Objekt verwandelt sich nur unter bestimmten Bedingungen.

Ein Objekt kann in dem Moment, in dem es sich ändert, eine Nachricht „MsgSelf“ an sich selber schicken. Mit diesem Mechanismus lassen sich Folgen von Ereignissen realisieren (siehe Abbildung 8).

Das Objekt A erhält die Nachricht b, wandelt sich in Objekt B und schickt an sich selbst die Nachricht c. Daraufhin wandelt es sich in C und sendet die Nachricht d an sich selbst.

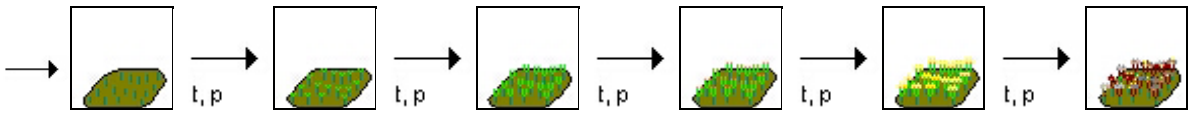

Abbildung 8: Folgen von Ereignissen, Zum Beispiel bei der Aussaat.

Über „MsgOther“ können Nachrichten an andere Objekte gesandt werden. Abbildung 9 zeigt drei Objekte, die über die zeit verändert werden. Das mittlere Objekt verändert sich gemäß den Angaben in „MsgSelf“, verwandelt aber das obere und untere Objekt gemäss den Angaben in MsgOther, die sich dann selber wieder verändern.

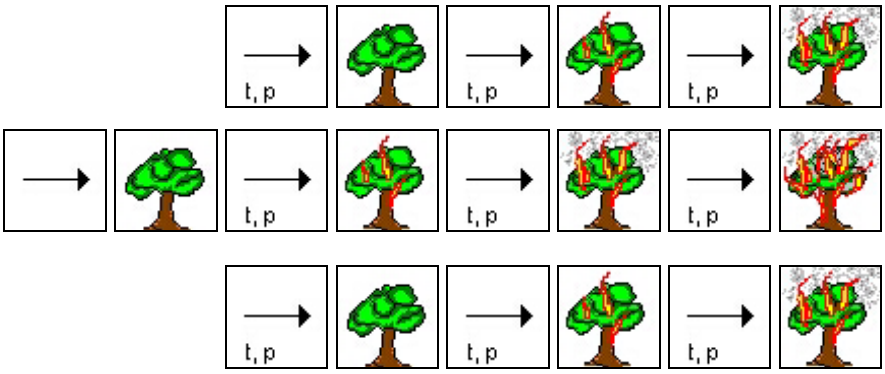

Abbildung 9: Die Verwendung von MsgSelf und MsgOther.

Im obigen Beispiel wird die Nachricht 'Anzünden' an umliegende Objekte gesandt (siehe mittlere Zeile in Abbildung 9). Dieser Nachricht kann ein Parameter beigefügt werden, der den Radius bestimmt, in dem diese Nachricht an andere Objekte gesandt wird ('Anzünden 150'). Wird dieser Parameter nicht angegeben, oder hat er den Wert 0, wird die Nachricht an alle anderen Objekte gesandt. Über diesen Mechanismus lassen sich sowohl die Ausbreitung von Feuer, als auch beispielsweise von Viren, gestalten.

Die boolesche Variable „AutoRepeat“ bestimmt, ob eine Nachricht, die in MsgSelf definiert ist, automatisch wiederholt wird.

„AutoStart“ und „AutoStartRandom“ sorgen dafür das eine Nachrichtricht bereits beim Starten des Programms gesendet wird. Die erste Variable sorgt für den Start einer Nachricht nach Ablauf der in Dauer festgelegten Zeit. Die zweite Variable gewinnt aus der Dauer einen Zufallswert, der maximal den in Dauer beschriebenen Wert annimmt.

Machen wir uns das am Beispiel einer fortlaufenden Bewegung deutlich: die Nachricht 'Move' ist eine spezielle Nachricht, auf deren Empfang hin sich ein Objekt um eine bestimmte Schrittweite in eine bestimmte Richtung bewegt. Um eine fortwährende Bewegung zu erzielen, muss diese Nachricht wiederholt gesandt werden (AutoRepeat = 1). Damit sich das Objekt vom Start des Programms an bewegt, sollten AutoStart oder AutoStartRandom auf den Wert 1 gesetzt werden.

Mit den Nachrichten 'Start' und 'Stop' läßt sich die Bewegung wieder aufnehmen oder anhalten. 'Step' sorgt für einen Einzelschritt des Objektes. 'Vorwärts' und 'Rückwärts' schalten in den Vorwärts- oder Rückwärtsgang. 'XMove' wirkt sich wie 'Step' aus, ohne dass die Umgebungsbedingungen berücksichtigt wird. 'FMove' macht dasselbe mit 10-facher Geschwindigkeit. Die letzten beiden Nachrichten sind ausschliesslich für Testzwecke definiert, die Agenten können sie nicht verwenden.

Die Nachrichten 'Left', 'Right', 'Up' und 'Down' sorgen für eine entsprechende Drehung des Objektes. Die Blickrichtung ist mit der Bewegungsrichtung identisch.

Mit der Nachricht 'PushGoal' werden Zielkoordinaten an ein Objekt gesendet. Diese Nachricht wird für bewegliche Objekte verwendet, die sich in der Umwelt nicht zufällig bewegen sollen, aber auch nicht von einem Agenten gesteuert werden. Diese Nachricht benötigt Parameter: 'PushGoal x y [d][*]'. Dies sind die Zielkoordinaten x und y, und ein optionaler Wert d, der angibt, wie groß der Abstand zum Ziel maximal sein darf, damit es als erreicht gilt. Ist der letzte Wert nicht angegeben, wird von einer Distanz von 50.0 Einheiten ausgegangen. Befindet sich in der Parameterliste ein '*', so wird das Ziel nicht angestrebt, sondern das Objekt bewegt sich von den Zielkoordinaten weg. Es können mehrere Ziele in einem Stack verwaltet werden. Aktuell ist immer das zuletzt angegebene Ziel. Mit 'PopGoal' wird das aktuelle Ziel von diesem Stack entfernt.

Um sich auf ein Ziel hinzubewegen oder auszurichten braucht ein Objekt noch eine Nachricht 'Goal'. Diese sollte, ähnlich wie die Nachricht 'Move', regelmäßig wiederholt werden. Sie sorgt für die Ausrichtung des Objekts auf den aktuellen Zielpunkt.

Die Nachricht 'New' erzeugt neue Objekte. Sind keine Parameter angegeben, so wird ein Objekt mit dem Objekttyp des sendenden Objektes erzeugt. Sind Namen von Objekttypen als Parameter angegeben, so werden Objekte des jeweiligen Typs erzeugt. Die Plazierung des neuen Objektes richtet sich nach der Umgebung. Wenn möglich, wird es in Blickrichtung mit dem Mindestabstand erzeugt. Ansonsten wird ein Platz gesucht, der möglichst nah am erzeugenden Objekt, nicht aber unbedingt in dessen Blickrichtung liegt.

Objekte verfügen über einen Behälter, in den sie andere Objekte mit der Nachricht 'Take' aufnehmen können.

Auch diese Nachricht benötigt unter Umständen Parameter: 'Take [ObjektTyp1 [ObjektTyp2]]'. Objekttyp1 gibt an, als welches Objekt das zu nehmende Objekt in den Behälter wandert, Objekttyp2 gibt an, in welches Objekt sich das zu nehmende Objekt wandelt.

Im *einfachsten* Fall bekommt ein Objekt die Nachricht 'Take' und wandert in den Behälter des sendenden Objektes (siehe Abbildung 10).

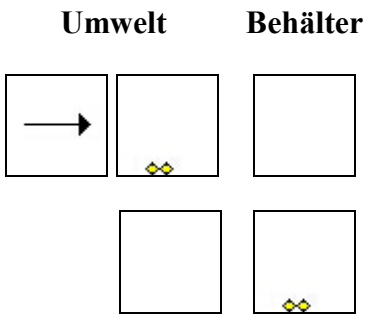

Abbildung 10: Behälter I.

Ein Objekt kann bei der Aufnahme auch gewandelt werden. Hier wird ein Stamm als Bretter aufgenommen: 'Take Bretter' (siehe Abbildung 11).

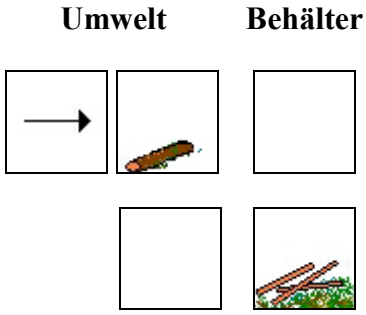

Abbildung 11: Behälter II.

Im nächsten Fall wird auch das Objekt in der Umwelt gewandelt: 'Take Nukleos Lava' (siehe Abbildung 12).

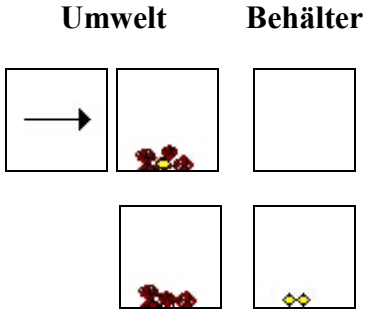

Abbildung 12: Behälter III.

Mit der Nachricht 'Give Objekttyp' wird ein Objekt aus dem Behälter wieder abgelegt. Die Plazierung erfolgt wie unter 'New' geschildert.

Will man erreichen, dass bestimmte Aktionen nach Ablauf einer festgelegten Zeit gestartet werden ('Events'), so genügt es, mit MsgSelf diese Aktion nach einer bestimmten Zeit (Auto-Start) auszulösen. Auf diese Weise können vom Programmierer (oder besser: Spielentwickler) Ereignisse vor dem Spielstart definiert werden, die die Agenten als „plötzlich“ eintretende Katastrophen erleben; wie z.B. der Ausbruch eines Waldbrandes nach 30 Minuten Spielzeit, den die handelnden Agenten dann z.B. kooperativ bewältigen müssen.

6. Ausblick

Diese Umwelt wird von uns eingesetzt werden, a) um sozionische Forschungsfragen in Bezug auf Gesellschaftsbildungsprozesse in Gruppen künstlicher autonomer Agenten zu untersuchen, b) um Gruppenprozesse von Gruppen menschlicher Versuchspersonen, die diese Umwelt als „Abenteuerspiel“ vorgelegt bekommen, zu untersuchen und c) um hybride Gemeinschaften in dieser Umwelt miteinander agieren zu lassen.

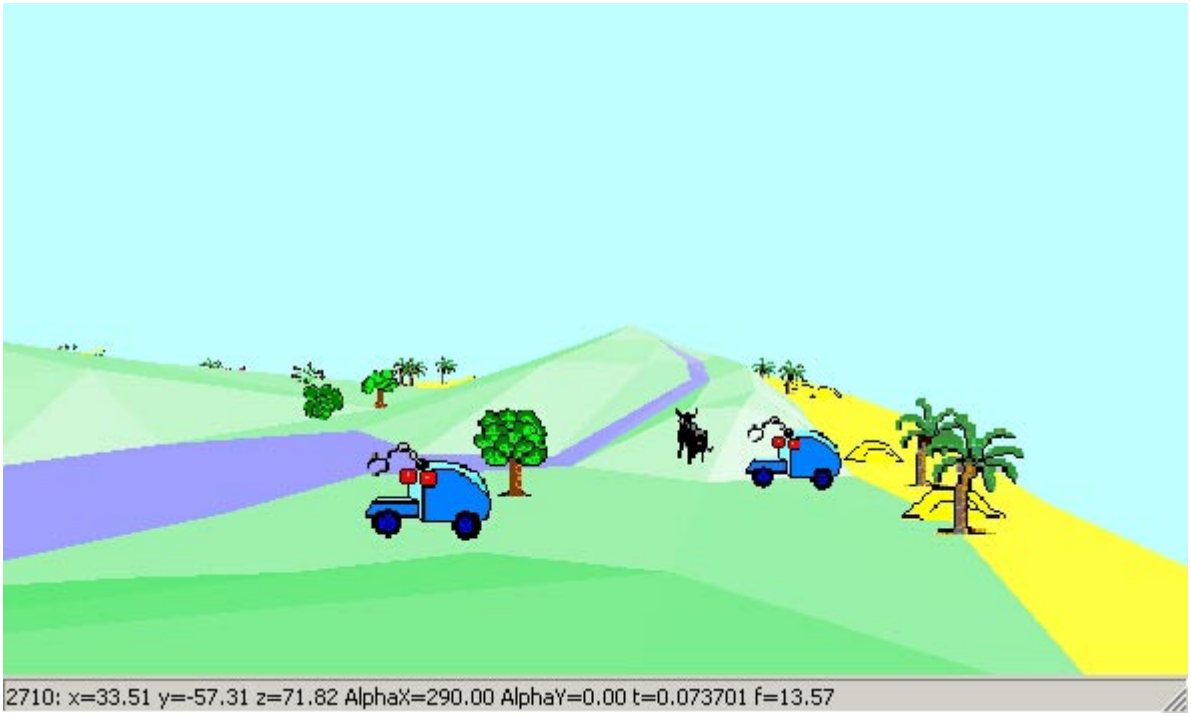

Abbildung 13: Zwei Agenten im Blickfeld einer Versuchsperson bzw. eines PSIs.

Abbildung 13 zeigt beispielhaft die Ansicht eines autonomen Agenten in seiner Umwelt. Neben einigen Objekten sieht er zwei weitere Agenten, von denen er nicht weiss, ob es sich um menschliche oder künstliche Agenten handelt.

7. Literatur

Detje, F. (2000a): Comparison of the PSI-theory with human behaviour in a complex task. In Niels Taatgen & Jans Aasman (Hrsg.): *Proceedings of the Third International Conference on Cognitive Modeling*. Veenendaal: Universal Press. S.86-93.

Detje, F. (2000b): ”Insel”. Dokumentation – Versuche – Ergebnisse. Bamberg: *Institut für Theoretische Psychologie, Memoramdum Nr. 39*.

Dörner, D. (1989): *Die Logik des Mißlingens*. Reinbek: Rohwolt.

Dörner, D. (1999): *Bauplan für eine Seele*. Reinbek: Rohwolt.

Dörner, D. (2000): The Simulation of Extreme Forms of Behaviour. In N. Taatgen & J. Aasman (Hrsg.): *Proceedings of the Third International Conference on Cognitive Modeling*. Veenendaal: Universal Press, S.94-99.

Dörner, D. & Gerdes, J. (1998-2001): *Psi – Eine psychologische Theorie als Computerprogramm*. http://www.uni-bamberg.de/ppp/insttheopsy/psi-software.html.

Gerdes, J. & Dörner, D. (2001): *Psi 3D – Ein System für die Simulation verschiedener, komplexer und dynamischer Umwelten*. http:// www.uni-bamberg.de/ppp/insttheopsy/psi3D.html

Turing, A. (1950): Computing Machinery and Intelligence. *Mind, 59*, S.433-460.