

# INFORMATIK UND UNTERRICHT



# INFORMATIQUE ET ENSEIGNEMENT

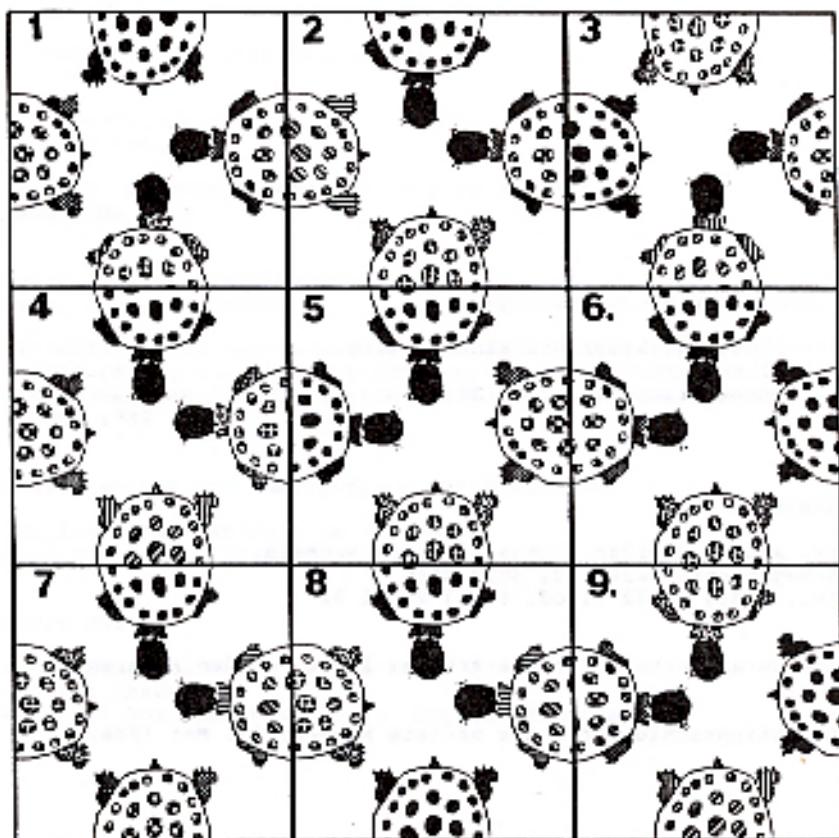
Informationebulletin  
zur Informatik an höheren  
Mittelschulen  
des Kantons Bern

Bulletin d'information  
concernant l'informatique dans  
les écoles moyennes supérieures  
du canton de Berne

Nummer 3, Februar 1986

Nunéro 3, février 1986

---



**Zum Titelblatt: Kartenspiel**

Man kopiere das Titelblatt und schneide die Karten aus. Dann ordne man sie derart in einem 3x3-Gitter an, dass zusammenstoßende Schildkrötenhälfte aufeinanderpassen. (Es gibt vier verschiedene Schildkrötenarten: volle, punktierte, karierte und gestrichelte.)

Wer das Probieren missfällt, der schreibe ein Programm. Nur dieser wird die Frage beantworten können, wieviel Lösungen das Spiel zulässt!

Lösung mit Kommentar im nächsten Bulletin.

**Herausgeberin und Bezugsquelle:**

Erziehungsdirektion des Kantons Bern  
Abteilung Unterricht  
Sulgeneggstrasse 70, 3005 Bern

**Redaktion:**

Dr. Aegidius Plüss, Gymnasium Bern-Neufeld,  
Bremgartenstrasse 133, 3012 Bern  
Tel. (031) 23 72 11 od. (031) 23 81 32

Die Verantwortung für die Artikel liegt bei den Autoren.

Redaktionsschluss für die nächste Nummer: 31. Mai 1986.

## Rekursive Lösung eines Kartenspiels in LOGO

Hans Salvisberg, Universität Bern

Die hier beschriebene Lösung des Puzzles aus dem Bulletin Nr. 3 (Titelblatt) verwendet einen sehr allgemeinen Suchalgorithmus, der unter dem Namen Backtracking bekannt ist. Er funktioniert genau so, wie man sich eine systematische Suche von Hand vorstellen könnte. Die folgenden Anweisungen sind ein bisschen kompliziert, deshalb nehmen Sie am besten gleich Ihr Kartenspiel zur Hand:

1. Links legen Sie immer die Karten hin, die Sie noch nicht probiert haben (also zu Beginn alle! und zwar aufsteigend sortiert, d.h. diejenigen mit der kleinsten Nummer oben und die Nummern immer in der linken oberen Ecke; dies ist der Vorrat). In der Mitte suchen Sie die Lösung, die Sie zeilenweise von links nach rechts und von oben nach unten aufbauen, in der Reihenfolge, in der die Karten auf der Vorlage nummeriert sind. Rechts "in den Papierkorb" legen Sie die Karten, die nicht passen.
2. Sie nehmen die oberste Karte aus dem Vorrat und probieren, ob sie passt. (Die erste passt natürlich immer.) Wenn sie passt, fahren Sie bei 3. weiter. Wenn sie nicht passt, drehen Sie sie um 90° und probieren wieder, bis zu dreimal. Passt die Karte nicht, so legen Sie sie in den Papierkorb und versuchen es mit der nächsten, bis Sie eine passende gefunden haben. Passt überhaupt keine, fahren Sie bei 4. weiter.
3. Vergleichen Sie mit der Ausgangslage zu 2., so haben Sie jetzt eine Karte mehr gelegt. Für den nächsten Platz sind nun wieder alle noch freien Karten im Vorrat, auch diejenigen aus dem Papierkorb. (Aufsteigend sortieren!) Falls noch nicht alle neun Karten gelegt sind, fahren Sie rekursiv bei 2. weiter, sonst sind Sie fertig.  
 ("Rekursiv" heisst hier in wesentlichen, dass Sie sich die aktuelle Situation merken, um später einmal genau hier weiterfahren zu können, und dass Sie nun die gleichen Schritte nochmals ausführen, wobei Sie jedesmal ein bisschen näher an die Lösung herankommen. Im allgemeinen Backtracking-Algorithmus muss man sich merken, welche Karten noch im Vorrat und welche schon im Papierkorb sind, und wieviel Mal man die eben gelegte Karte bereits gedreht hat. Da aber die Karten immer aufsteigend sortiert sind, ist es klar, dass, wenn Sie beispielsweise die Karte 5 gelegt haben, die Karten 1 bis 4 entweder in der Lösung eingebaut oder aber im Papierkorb sind. Außerdem wissen Sie auch, wie oft Sie die Karte schon gedreht haben, da Sie zu Beginn die Zahl immer links oben haben.)

4. Wenn Sie hier ankommen, haben Sie alle noch freien Karten ausprobiert und in den Papierkorb geworfen. Also sind Sie mit der zuletzt gelegten Karte (nennen wir sie X) in eine Sackgasse geraten. Sie gehen in der Rekursion eine Stufe zurück, d.h. Sie stellen die Situation wieder her, wie sie war, als Sie X legten, und befinden sich nun mitten in Punkt 2. Diese Karte X, von der Sie glaubten, sie passe, war nämlich doch nicht die richtige. (zu 2.: Falle Sie X noch nicht dreimal gedreht haben, versuchen Sie es weiter mit X, sonst legen Sie X in den Papierkorb und nehmen die nächste Karte.)
- +. Wenn Sie nicht nur eine, sondern alle Lösungen finden wollen, hören Sie in 2. einfach nicht auf, sondern fahren bei 4. weiter.

An meisten Mühe, abgesehen von der schier unendlichen Dauer der Suche, macht uns Menschen wohl die Rekursion in den Punkten 2. und 4., die sich nur schwer in unser sequentielles Denkschema pressen lässt. Wer sie nicht kennt, dem ist sie zuerst etwas unheimlich – man versteht nicht recht, wieso das funktionieren soll. Doch mit der Zeit gewinnt man Vertrauen und lernt die Eleganz der rekursiven Problemlösung schätzen.

#### Implementation in LOGO

Die Programmiersprache LOGO eignet sich hervorragend zur Implementation von rekursiven Algorithmen. Da es nur dynamische Datenstrukturen gibt, sind diese sehr gut unterstützt (hier Kartonstapel, die wachsen und abnehmen können und manchmal auch zusammengefügt werden). (Die Nummern in eckigen Klammern verweisen im folgenden auf die entsprechenden Stellen in der Programmliste.)

Eine wichtige Vorentscheidung fällt bei der Wahl der Datenstruktur: Die Liste :schildkroeten [1] dient als Eingabe für das Programm und enthält neun Unterlisten, für jede Karte eine. Die Reihenfolge entspricht genau der Zeichnung im Bulletin Nr. 3. Jede Teilliste enthält die Nummer der betreffenden Karte, die Anzahl 90°-Drehungen, die damit ausgeführt wurden (hier überall 0) und eine weitere Unterliste, die die Schildkrötentäte auf dieser Karte beschreibt. Beispiel: Auf der zweiten Karte befindet sich oben der 1. (vordere) Teil einer ausgefüllten Schildkröte, sowie im Uhrzeigersinn der 1. Teil einer kartierten, der 2. (hintere) Teil einer gestupften und der 2. Teil einer gestrichelten.

Das Hauptprogramm suche [2] wird mit suche :schildkroeten aufgerufen und ruft im wesentlichen die Prozedur suchen [3,4] auf, mit der Eingabelliste und zwei leeren Listen.

suchen [4] hat drei Parameter, die alle Listen der Art von :schildkroeten sind; :vorrat enthält die Karten, die noch nicht

probiert wurden, gelegt die angehende Lösung und vorbei die Karten im Papierkorb. Zuerst wird geprüft, ob alle neun Karten gelegt wurden [6], dann wäre eine Lösung gefunden und könnte ausgegeben werden [7]. Dann wird geprüft, ob noch Karten im Vorrat sind [8]. Wenn nicht, geht die Programmausführung um eine Rekursionsstufe zurück [12/14]. Ein derartiges Abbruchkriterium [8] muss in jeder rekursiven Prozedur enthalten sein.

In der aktuellen Ausgabe von suchen wird versucht, die erste Karte aus vorrat zu legen: Dessen Beschreibung wird der Variablen karte zugewiesen [9]. Die LOCAL-Anweisung [5] sorgt dafür, dass karte auf jeder Stufe der Rekursion separat erhalten bleibt. Kann karte gelegt werden [11], so beginnt suchen wieder von vorne, mit dem neuen vorrat bestehend aus den Karten im Papierkorb (vorbei) und dem alten vorrat. Zu der Liste gelegt mit den erfolgreich gelegten Karten wird die karte hinzugefügt; probiert wurde noch keine Karte, also ist vorbei leer. - Nach der Rückkehr aus der Rekursion wird die karte um 90° gedreht [13,26].

Die Anweisungsliste der REPEAT-Anweisung wird viermal ausgeführt, um alle möglichen Drehrichtungen zu erhalten, außer wenn die Karte in Zentrum gelegt wird [10]. Dank diesem Trick findet das Programm nur die echt verschiedenen Lösungen, und alle, die daraus durch Drehung entstehen, werden unterdrückt.

Nachdem mit dieser Karte alle Möglichkeiten durchgespielt wurden, ruft suchen sich selbst wieder auf, wobei karte aus vorrat entfernt und in vorbei abgelegt wird. Hier handelt es sich nur der Form nach um eine Rekursion, denn die aktuelle Situation interessiert ja nach der Rückkehr nicht mehr: es bleibt nur noch die END-Anweisung. Obwohl eine (Endlos-)Schleife den gleichen Effekt hätte, ist es in LOGO üblich, die elegantere, rekursive Formulierung zu verwenden. Gute LOGO-Implementierungen sind ohnehin in der Lage, diesen Spezialfall zu erkennen und durch eine Iteration zu ersetzen.

Die Funktion testkarte [15] prüft, ob karte zu gelegt passt. Wenn noch gar keine Karte gelegt ist, passt sie sowieso [16]. Nur wenn nicht genau drei oder genau sechs Karten gelegt sind [17], muss sie gegen links passen [18]. Wenn mindestens drei Karten schon gelegt sind [19], muss sie auch gegen oben passen [20]. Ist sie bei keinem der Tests durchgefallen, so passt sie [21].

Die Funktion test2 [22] erhält als Argumente die Beschreibungen zweier Schildkrötenhilflien, z.B. f2 und f1, und prüft, ob diese zusammenpassen. (Hier wäre das der Fall.)

Die mod-Funktion [23], die den Divisionsrest liefert, wird natürlich rekursiv programmiert.

Die Funktion drehe [24] führt in einem recht komplizierten Ausdruck die Operation einer 90°-Drehung einer Karte durch.

Bemerkenswert ist hier, dass LOGO als Funktionswert eine ganze Liste abgeben kann.

Die Prozedur loesung [25] gibt einen Kommentar aus und ruft die Prozedur prloesung [26] auf, die eine Lösung mit einer Karte pro Zeile ausgibt. Typisch für LOGO ist die rekursive Verarbeitung der Liste.

(Bemerkung: Da LOGO eine interpretierte Sprache ist, läuft das Programm recht lange. Sie sollten mit einigen Stunden rechnen!)

Anmerkung der Red.:

Das publizierte Kartenspiel besitzt (abgesehen von der Rotationssymmetrie) folgende 12 Lösungen:

(zeilenweise (durch Komma abgetrennt): Kartennummer, anschliessend Lage der Nummer auf der Karte: oben (o), rechts (r), unten (u), links (l))

1. 112r8r,314r9u,5r6u7r
2. 2r3o5u,4r9u1o,6u7r8r
3. 2r3o5u,4r9u1o,7r8r6u
4. 2r3o5u,4r9u1o,6r6u7r
5. 2r8r6u,4r9u1o,5u3o7r
6. 2r8r6u,4r9u1o,7r3o5u
7. 2u7u4u,3r913u,5llz6l
8. 2u7u4u,8u913r,6llt5l
9. 2u8u4u,61913r,7u1t5l
10. 4u2u7u,3r918u,5llr6l
11. 4u2u7u,8u913r,6llz5l
12. 4u2u8u,61913r,7u1r5l

#### LOGO Programmcode

```
MAKE "schildkröten"
[t1 0 [f2 k1 t1 k2]] [2 0 [f1 k1 t2 s2]] [3 0 [t2 k1 s1 f2]] >
[4 0 [f1 t1 s2 k2]] [5 0 [f1 k2 t2 t1]] [6 0 [f1 f2 t2 k1]] >
[7 0 [f1 s1 k2 t2]] [8 0 [f1 s1 f2 t2]] [9 0 [t1 f2 s2 k1]]]
```

```
TO suche :s
  PR [Ich beginne zu suchen]
  suchen :s []
  PR []
  PR [Das Programm ist zu Ende]
END
```

```

TO suchen* vorrat :gelegt vorbei
LOCAL "karte"
IF EQUALP 9 COUNT :gelegt * *
  lösung :gelegt ?
IF EQUALP 0 COUNT vorrat * *
  [STOP]
MAKE "karte FIRST vorrat *
REPEAT (IF EQUALP 4 COUNT :gelegt [1] [4] *) *
  [( IF testkarte :gelegt "karte " *
    [suchen BE vorbei IF vorrat LPUT :karte :gelegt [1] " ] )*
    MAKE "karte drehe :karte " )
suchen BE vorrat :gelegt LPUT FIRST vorrat vorbei "
END

TO testkarte* :gelegt :karte
IF EMPTYP :gelegt * *
  [OP "TRUE"]
IF NOT EQUALP 0 mod COUNT :gelegt 3 " *
  [(IF NOT test2 LABT LAST :karte ITEM 2 LABT :gelegt " *
    [OP "FALBE"])
  IF (COUNT :gelegt) > 2 " *
    [(IF NOT test2 FIRST LABT :karte *
      ITEM 3 LABT ITEM ((COUNT :gelegt) - 2) :gelegt " *
        [OP "FALBE"])
  OP "TRUE "
END

TO test2* :hi :h2
IF NOT EQUALP FIRST :hi FIRST :h2 *
  [OP "FALSE"] *
  [OP NOT EQUALP LABT :hi LABT :h2]
END

TO mod** :a :b
IF :a < :b *
  [OP :a] *
  [OP mod (:a - :b) :b]
END

TO drehe** :karte
OP LPUT BE BE LABT :karte FIRST LABT :karte *
  BE FIRST :karte mod (1 + ITEM 2 :karte) 4
END

TO lösung** :i1
PR []
PR [eine Lösung ist: ]
prlöesung :=1
PR []
END

TO prlöesung** :liste
IF EMPTYP :liste [STOP]
PR FIRST :liste
prlöesung BE :liste
END

```



