COMPLETE ANALYSIS OF MISÈRE GAMES

In this chapter we will prove a theorem which will allow us to extend the information from the generalized genus algorithm to a complete solution of certain misère games. We will conclude by applying this method to several particular games.

Guy and Smith [10] have shown that for certain nimlike games in normal play, once we have verified that the nim values of the various heaps repeat to a certain point, that they will continue to repeat, and we can claim a complete solution. I will review this theorem here, and apply the same technique to games with misdre play.

Theorem **V** (Recurrence of normal Grundy values) (Guy and Smith)

For some octal game (i.e., a nim-like game where the rules allow the players to remove some counters from a heap, and/or to split the remaining heap into two smaller heaps), denote a heap of n counters by H,. Let r be the maximum number of counters which may be removed from a heap. We insist that r be finite. If there exist some integers m and s such that for any i with $m < i \leq 2m + s + r$, $H_i = H_{i+s}$, then for $i > m$, $H_i = H_{i+s}$.

Proof. Suppose the result is not true: then let t be the smallest integer $(t > m)$ such that $H_t \neq H_{t+s}$. We know that $t > 2m + s + r$.

For any $j \leq r$, if the rules allow a player to remove j counters without splitting a heap, then the options of H_t include H_{t-j} and the options of H_{t+s} include H_{t+s-j}. Since $m < t - j < t$, $H_{t-j} = H_{t+s-j}$, and these options are equal.

If the rules allow a player to remove j counters without splitting a heap, then the options of H_t include

$$H_n + H_{t-j-n} \qquad 1 \leq n \leq \left\lfloor \frac{t-j}{2} \right\rfloor$$

and the options of H_{t+s} include

$$H_n + H_{t+s-j-n} \qquad 1 \leq n \leq \left\lfloor \frac{t-j+s}{2} \right\rfloor$$

For $n \leq \left\lfloor \frac{t-j}{2} \right\rfloor$, we have $t - j - n \geq \left\lceil \frac{t-j}{2} \right\rceil > m$, and $t - j - n < t$, so we have that $H_{t-j-n} = H_{t+s-j-n}$, and the options of H_t are included in those of H_{t+s}.

For $n \leq \left\lfloor \frac{t-j+s}{2} \right\rfloor$, we have

$$\begin{aligned} t + s - j - n &\geq t + s - j - \left\lfloor \frac{t+s-j}{2} \right\rfloor \\ &= \left\lceil \frac{t+s-j}{2} \right\rceil \\ &\geq \left\lceil \frac{t+s-r}{2} \right\rceil \\ &> \left\lceil \frac{2m+2s}{2} \right\rceil = m + s \end{aligned}$$

We have a strict inequality on the last line because $t > 2m + s + r$ and $\frac{2m+2s}{2}$ is an integer.

Now $m < t - j - n < t$, so $H_{t-j-n} = H_{t+s-j-n}$ and for every option of H_{t+s} there is an equal option of H_t.

For $0 \leq j \leq r$ the options discussed above are all the options of H_t and H_{t+s}, so we have $H_t = H_{t+s}$. ■

Now we adapt this theorem for miskre play. We could use this theorem verbatim for miskre play, but it requires that we find several games to be *equal.* This very rarely happens with miskre games, so this would not be of much use. For this reason we will work this out using $\equiv$ instead of $=$.

Theorem VI. (recurrence of miskre values).

For some octal game, let H_n be the heap of n counters, and let r be the maximum number of counters (r finite) which may be removed from a heap.

If there exist integers m and s such that for i with $m < i \leq 2m + s + r$, $H_i \equiv H_{i+s}$ ((heaps up to $2m + 2s + r$)), then for $i > m$, $H_i \equiv H_{i+s}$ ((all heaps)).

Proof. Let t be the smallest integer ($t > m$) such that

$$H_t \not\equiv H_{t+s} \qquad (\langle \text{heaps up to } s + t \rangle)$$

We know that $t > 2m + s + r$. The argument of Theorem V shows that for any option H'_t we can find an option H'_{t+s} such that

$$H'_t \equiv H'_{t+s} \qquad ((\text{heaps up to } s + t - 1))$$

and vice versa.

Now I claim that

$$n \cdot H_t \equiv n \cdot H_{t+s} \qquad ((\text{heaps up to } s + t - 1)) \text{for all } n \geq 0$$

If not, let k be the least integer such that

$$k \cdot H_t \not\equiv k \cdot H_{t+s} \qquad (\langle \text{heaps up to } s + t - 1 \rangle)$$

That is, there is some s $\in$ (heaps up to $s + t - 1$) such that

$$w + k \cdot H_t \not\sim w + k \cdot H_{t+s}$$

Choose w simplest with this property. The options of the left hand side are

$$w' + k \cdot H_t$$

and

$$w + H'_t + (k-1) \cdot H_t$$

$w' + k \cdot H_t \sim w' + k \cdot H_{s+t}$ by minimality of w.

There exists a H'_{s+t} such that

$$w + H'_t + (k-1) \cdot H_t \sim w + H'_{s+t} + (k-1) \cdot H_t$$

since $w + (k-1) \cdot H_t \in \langle \text{heaps up to } s+t-1 \rangle$.

Also,

$$w + H'_{s+t} + (k-1) \cdot h_t \sim w + H'_{s+t} + (k-1) \cdot H_{s+t}$$

by minimality of k.

But the options of $w + k \cdot H_{s+t}$ are exactly

$$w' + k \cdot H_{s+t}$$

and

$$w + H'_{s+t} + (k-1) \cdot H_{s+t}$$

So we may conclude that $w + k \cdot H_t \sim w + k \cdot H_{s+t}$, and hence, we conclude (1). But (1) is clearly equivalent to

$$H_t \equiv H_{s+t} \qquad (\langle \text{heaps up to } s+t \rangle)$$

This contradicts our choice of t. So we have

$$H_i \equiv H_{i+s} \qquad (\langle \text{heaps up to } i+s \rangle) \qquad \text{all } i > m \tag{2}$$

Now for any finite sum of heaps with at least one heap of more that $m+s$ counters, let $b+s$ be the size of the largest heap. From (2) we have that

$$H_b \equiv H_{b+s} \qquad (\langle \text{heaps up to } b+s \rangle)$$

This tells us that we can replace all heaps of size $b+s$ by heaps of size b. We may repeat this on all heaps larger that $m+s$ to tell us that

$$H_i \equiv H_{i+s} \qquad (\langle \text{all heaps} \rangle) \text{ all } i > m$$

■

To use this theorem, we need to know the many-dimensional genus array of 0, in which the various dimensions correspond to the various heaps of some octal game. Fortunately, this is exactly the sort of information which is supplied by the algorithm based on theorem III.

For the game of Guiles (octal code .15) which we discussed earlier, even if we neglect the irregularities at 47, 49, and 59, we still must verify sums of heaps up to 64 (the last irregularity is at 21, period is 10, 2 counters may be removed in one move). This requires that we compute roughly 3^{41} genus sequences. This is infeasible even with the aid of a computer.

For the games .75, .72, .54, and .53 this method is feasible. The solutions to these games appear in the catalogue in the next section. The 'proofs' that these solutions are correct consist of several pages of computer output verifying that the hypotheses of theorem VI have been satisfied. In all cases, the computation was carried out by hand once the computer had verified enough patterns to make this feasible.

A CATALOGUE OF OCTAL GAMES

The following table records the status of our knowledge of two-digit octal games. This information is condensed from chapter 13 of *Winning Ways* and the analysis presented in this work.

	0	1	2	3	4	5	6	7
.0	nim	nim	nim	nim	mess	nim	mess	mess
.1	nim	nim	nim	nim	mess	mess	mess	mess
.2	nim	nim	nim	nim	nim	nim	done	copy
.3	nim	tame	nim	nim	done	big	mess	mess
.4	copy	copy	copy	copy	copy	mess	copy	copy
.5	nim	nim	tame	done	done	nim	tame	copy
.6	copy	copy	copy	copy	mess	copy	copy	copy
.7	nim	done	done	tame	mess	done	mess	mess
4.	nim	nim	nim	tame	copy	nim	copy	done

NIM - Every position is misère equal to a nim position, hence we can use the theory for normal play to determine the outcome.

TAME- Every position is tame, as defined in WW chapter 13.

COPY- The genus information of these games appears in some other game: according to the copy table on page 44. All examples of COPY are examples of concepts defined by Guy and Smith [10].

DONE- A complete solution has been obtained for several games, as described on pages 44–47.

BIG - For the game 0.35 I have done computations too extensive for the format of appendix 11, but have obtained no complete solution. A partial solution appears on page 48.

MESS - The genus sequences show no obvious pattern which would allow us to conjecture a complete solution. In these cases, we must settle for an incomplete analysis as presented in appendix 11.

— DONE —

For many of the octal games: the computed outcomes suggested patterns which have been proven to exist. The case 4.7 was treated in detail in the last chapter, and the cases 0.75, 0.72, 0.54 and 0.53 rely upon theorem VI. The solutions for 0.34 and 0.71 are due to Jim Flanigan, and are reprinted here from *Winning Ways*.

$.27(n) = .26(n)$ $.60(n) = .37(n-1)$

$.40(n) = .07(n-1)$ $.61(n) = .37(n-1)$

$.41(n) = .17(n-1)$ $.62(n) = .37(n-1)$

$.42(n) = .07(n)$ $.63(n) = .37(n-1)$

$.43(n) = .17(n-1)$ $.65(n) = .64(n)$

$.44(2n+2) = .44(2n+1) = .77(n)$ $.66(n) = .64(n)$

$.46(2n+2) = .46(2n+1) = .77(n)$ $.67(n) = .64(n)$

$.47(n) = .45(n)$ $4.4(n) = .77(n-1)$

$.57(2n) = .57(2n-1) = 4.7(n)$ $4.6(n) = .77(n-1)$

A line of the form $.ab(n) = .cd(m)$ means that a heap of m counters played according to the rules of $.cd$ is equal in misère play to a heap of n counters played according to the rules of $.ab$.

Fig. 1 - The COPY table

The game *0.26*

We state the solution in terms of the *tameness* $t(n)$ and *wildness* $w(n)$ of a heap of n counters as given by the following table :

n =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
t(n) =	0	0	0	1	1	0	0	1	1	2	2	**5**	3	6	4	7	5	8	6	9	7	...
w(n) =	0	0	0	0	0	0	0	0	0	1	**1**	2	0	3	1	4	2	5	**3**	6	**4**	...

For large even n, $t(n) = \frac{n}{2} - 3$ and $w(n) = \frac{n}{2} - 6$. For large odd n, $t(n) = \frac{n-1}{2}$ and $w(n) = \frac{n-1}{2} - 3$.

For a .26 position, let $no \geq n_1 \geq \ldots > 0$ be the number of counters in each heap. We may assume that every heap has at least one counter, since no play is possible from a heap with no counters.

Let $\Delta = w(n_0) - \sum_{i>0} t(n_i)$

The outcome of a position is determined by the following rule:

If every heap has **1,** 2, **5** or 6 counters, then it is a Pposition iff and only if there is an odd number of heaps, of size 2 or 6

If there is any heap of some other size, then

$\mathbf{A} \leq 0 \Rightarrow o(K) = P$ iff ~~E and D~~ are both even — normal sum = 0

$\mathbf{A} = 1 \Rightarrow o(K) = N$

$\mathbf{A} \geq 2 \Rightarrow o(K) = P$ iff ~~E and D are both odd~~ — normal sum = 2

The proof of this is similar in nature to the proof given in the text for the game of *Knots* (4.7).

The game 0.34

In the following table, $a = \{:1, :4\}$, genus (1#20).

Each heap of 0.34 is a nimheap or one of the games $a = \{:1, :4\}$ and $a :1$ (genera (1#20) and (0#31) respectively), as given in the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	1	2	0	1	0	3	2	a	1	2	0	$a :1$

The last 8 values repeat indefinitely. We can pretend that $a + a = 0$.

The game 0.53

For positions in which at least two heaps have 6 or more counters, we can pretend that the various heaps of .53 equal adders or nimheaps according to the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
:0	:1	:1	:2	:2	:1	:2	:2	:2	*4	:4	:5	:2	:2	:5	:5	:2	:2	*4	:5

We can pretend that the last 9 entries repeat indefinitely.

For positions in which at most one heap has 6 or more counters, we can use the blurry genus table in appendix I.

The game 0.54

For positions in which at least two heaps have 9 or more counters, we can pretend that the various heaps of .54 equal adders or nimheaps according to the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
:0	:1	:0	:0	:2	:2	:2	*4	:1	:5	:5	:2	:2	:2	*4	:5

We can pretend that the last 7 entries repeat indefinitely.

For positions in which at most one heap has 9 or more counters, we can use the blurry genus table in appendix I.

The game 0.71

We can pretend that each heap of 0.71 is a nimheap or one of the games $r = \{:4,:5\}$,$s = \{:1,r:2\}$ (genera (2020) and (1#20) respectively) according to the following table:

0	1	2	3	4	5	6	7	8	
0	1	2	1	0	r	0	1	s	...

The last 6 values repeat indefinitely, and we can use the following table to compute the genus of sums of r and s:

+	0	r	$2r$	$3r, 5r, 7r, \ldots$	$4r, 6r, 8r, \ldots$
$0, 2s, 4s, \ldots$	1202	2020	0202	1313	0202
$s, 3s, 5s, \ldots$	1#20	2020	2020	1313	0202

The game 0.72

For positions in which at most one heap has more than 8 counters, the following adder/genus table gives the correct outcome:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
:0	:1	:0	:2	:3	:1	:0	:2	:3	#313	0202	2020	3131	#313	0202	1#20	2031

The last four values repeat indefinitely.

For more complicated sums, we may usually pretend that the heaps equal adders according to the following table:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	1	0	2	3	1	0	2	3	5	4	6	7	5	4	6	7	...

Again, the last four values repeat indefinitely.

The above table fails only for the cases displayed in the following table:

	+	9, 13	11, 15
c	16, 20, 24, ...	##20	#313

The game 0.75

In the following table, $r = \{:0, :2, :3, :4\}$, genus $(\#313)$.

We can pretend that the heaps of **.75** equal adders or r according to the following table :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
0	1	2	1	2	r	2	r	2	5	2	5	...

The last two values repeat indefinitely. We can pretend that $r + r = :4$.

The game of Knots (4.7)

We state the solution in terms of the ***tameness*** $t(n)$ and ***wildness*** $w(n)$ of a string of n knots as given by the following table:

$n =$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	...
$t(n) =$	**0**	**0**	**1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	...
$w(n) =$	0	0	0	0	**0**	**1**	**0**	**1**	**2**	**3**	...

For $n \geq 3$, $t(n) = n - 3$, for $n > 6$, $w(n) = n - 6$.

For a Knots position K, let $n_0 \geq n_1 \geq \ldots > 0$ be the number of knots in each string. We may assume that every string has at least one knot, since no play is possible form a string with no knots.

Let $\mathbf{A} = w(n_0) - \sum_{i>0} t(n_i)$

$E(K)$ = number of strings with an even number of knots

$D(K)$ = number of strings with an odd number of knots

The outcome of K is determined by the following rule:

If every string has length 1 or 3, then K is a P position if and only if there is an odd number of strings.

If there is any string of some other length, then

$\Delta \leq 0 \Rightarrow o(K) =$ P iff E and D are both even

$\mathbf{A} = 1 \Rightarrow o(K) = \mathcal{N}$

$\mathbf{A} \geq 2 \Rightarrow o(K) =$ P iff E and D are both odd

— BIG — The game 0.35

In this case, I have done computations too extensive for the format of appendix II, but no complete solution was obtained.

In the following discussion, let hn be a heap of h counters.

To find the blurry genus of any position in which no heap has more than 7 counters, use the following procedure:

First, reduce the number of heaps by the pretenses

$$h3 + h3 + h3 = h3$$
$$h6 + h6 + h6 = h6$$
$$h5 = :4$$

If there are at least two heaps of seven counters, or at least two heaps of four counters (regardless of the sizes of the other heaps), then apply the following pretenses:

$$h4 = h7 = :5$$

If there is exactly one heap of seven counters and exactly one heap of four counters, then apply the pretense

$$h4 + h7 = :2$$

If there is exactly one heap of 4 counters, and no heap of 7 counters, then replace $h4$ according to the following pretense table:

	0	$h3$	$h3 + h3$
0	:2	:5	:5
$h6$	:3	∞	:5
$h6 + h6$	:2	:5	:5

For example, in the sum $h6 + h6 + h4 + h3$, replace $h4$ by :5.

Any sum containing a term ∞ has genus (####).

If there is exactly one heap of size 7 and none of size 4, then use the following pretense table:

	0	$h3$	$h3 + h3$
0	:0	:5	:5
$h6$	:3	:0	:5
$h6 + h6$	:0	:5	:5

Finally, use the pretenses

$$h6 = :2$$
$$h3 = :4$$

The resulting sum of adders has the same blurry genus sequence as the original position.

APPENDIX I - GENUS SEQUENCES OF OCTAL GAMES

How to read Appendix I

Each column refers to a different octal game. The entry in row n is the blurry genus symbol of a heap of n counters, played according to the rules of the octal game associated with the column. A * before a genus symbol indicates that,the position is equal to a nimheap, although some positions without stars are also nimheaps.

Games for which a complete analysis (or a complete reduction to the normal case) appears in WW are not included.

heap size	0.04000	0.06000	Octal *game* 0.07000	0.14000	0.15000
0	* 0(1202)	* 0(1202)	* 0(1202)	* 0(1202)	* 0(1202)
1	* 0(1202)	* 0(1202)	* 0(1202)	* 1(0313)	* 1(0313)
2	* 0(1202)	* 0(1202)	* 1(0313)	* 0(1202)	* 1(0313)
3	* 0(1202)	* 1(0313)	* 1(0313)	* 0(1202)	* 0(1202)
4	* 1(0313)	* 1(0313)	* 2(2020)	* 1(0313)	* 1(0313)
5	* 1(0313)	* 2(2020)	* 0(1202)	* 0(1202)	* 1(0313)
6	* 1(0313)	* 2(2020)	* 3(3131)	* 2(2020)	* 2(2020)
7	* 2(2020)	* 0(1202)	* 1(0313)	* 1(0313)	* 2(2020)
8	* 2(2020)	* 3(3131)	* 1(0313)	* 2(2020)	* 1(0313)
9	* 0(1202)	* 1(0313)	* 0(1202)	* 2(2020)	* 2(2020)
10	* 3(3131)	* 1(0313)	3(1#31)	* 1(0313)	* 2(2020)
11	* 3(3131)	* 2(2020)	* 3(3131)	* 0(1202)	* 1(0313)
12	* 1(0313)	2(1#20)	2(0#20)	* 4(####)	* 1(0313)
13	* 1(0313)	3(1#31)	* 2(2020)	* 1(0313)	* 0(1202)
14	* 1(0313)	* 3(3131)	4(1###)	4(1###)	1(#313)
15	* 0(1202)	4(0###)	* 0(1202)	4(####)	1(#313)
16	4(1###)	4(0###)	5(0###)	1(3131)	* 2(2020)
17	3(1#31)	0(2020)	2(0#20)	2(1#20)	* 2(2020)
18	* 3(3131)	5(1###)	* 2(2020)	* 2(2020)	1(2020)
19	* 3(3131)	3(1#31)	3(1#31)	1(3131)	* 2(2020)
20	2(0#20)	3(3131)	* 3(3131)	2(1#20)	2(1#20)
21	* 2(2020)	4(0###)	0(0202)	0(0202)	1(#313)
22	* 2(2020)	2(0#20)	1(0313)	1(0313)	1(#313)
23	4(1###)	2(2020)	1(1313)	0(1202)	0(1202)
24	4(1###)	1(1313)	3(1#31)	4(0###)	1(#313)
25	0(1202)	1(1313)	0(3131)	1(3131)	1(#313)
26	5(0###)	3(3131)	2(0#20)	2(2020)	* 2(2020)
27	5(0###)	0(0202)	1(#313)	6(20##)	2(1#20)
28	2(0#20)	2(0#20)	1(1313)	1(3131)	1(2020)
29	* 2(2020)	2(2020)	0(1202)	6(####)	2(1#20)
30	* 2(2020)	1(1313)	4(0###)	4(0###)	2(1#20)
31	3(1#31)	1(1313)	5(0###)	1(3131)	1(1313)
32	3(3131)	4(####)	2(2020)	4(1###)	1(#313)
33	0(3131)	4(0###)	7(1###)	0(0202)	0(1202)
34	5(0###)	5(0###)	4(####)	1(3131)	1(#313)
35	0(0202)	5(####)	0(0202)	0(2020)	1(#313)
36	1(0313)	2(1#20)	1(0313)	2(2020)	2(2020)
37	1(1313)	6(1###)	1(1313)	1(3131)	2(1#20)
38	1(1313)	4(####)	2(1#20)	2(20##)	1(2020)
39	3(1#31)	7(0###)	0(3131)	6(0###)	2(1#20)
40	3(3131)	5(0###)	3(0#31)	1(1313)	2(1#20)
41	3(3131)	8(####)	1(#313)	4(####)	1(1313)
42	5(0#20)	1(1313)	1(1313)	4(0###)	1(#313)
43	6(2020)	2(1#20)	0(1202)	1(1313)	0(1202)
44	4(####)	2(2020)	3(3131)	4(####)	1(#313)
45	4(1###)	0(0202)	3(0#31)	6(####)	1(#313)
46	1(1313)	3(0#31)	2(2020)	1(31#3)	2(2020)
47	0(1202)	1(#313)	2(1#20)	2(20##)	2(1#20)
48	5(0###)	1(1313)	4(####)	2(####)	1(2020)
49	5(0###)			1(1313)	2(1#20)
50	6(0###)				2(1#20)

heap size	octal game 0.04000	0.06000	0.07000	0.14000	0.15000
51	6(####)				1(1313)
52	2(2020)				1(#313)
53	7(1#31)				0(1202)
54	7(####)				1(#313)
55	7(####)				1(#313)
56	8(0###)				2(2020)
57	0(0202)				2(2020)
58	1(0313)				1(2020)
59	9(1###)				2(2020)
60	2(1###)				2(2020)
61	7(1###)				1(1313)
62	2(####)				1(#313)
63	5(1313)				0(1202)
64	3(0#20)				1(#313)
65	3(1313)				
66	9(0###)				
67	0(1###)				
68	5(1###)				
69	4(1202)				
70	4(0###)				
71	8(0###)				
72	6(0###)				
73	6(0###)				

heap size	Octal game 0.16000	0.17000	0.26000	0.35000
0	* 0(1202)	* 0(1202)	* 0(12020202)	* 0(1202)
1	* 1(0313)	* 1(0313)	* 0(12020202)	* 1(0313)
2	* 0(1202)	* 1(0313)	* 1(03131313)	* 2(2020)
3	* 0(1202)	* 0(1202)	* 2(20202020)	0(0202)
4	* 1(0313)	* 2(2020)	* 3(31313131)	1(2020)
5	* 2(2020)	* 1(0313)	* 0(12020202)	0(0202)
6	* 2(2020)	* 3(3131)	* 1(03131313)	2(2020)
7	* 1(0313)	* 0(1202)	* 2(20202020)	1(0202)
8	* 4(####)	* 1(0313)	* 3(31313131)	2(2020)
9	* 0(1202)	* 1(0313)	0(#2020202)	0(0202)
10	* 1(0313)	3(1#31)	1(#3131313)	1(2020)
11	4(####)	* 2(2020)	2(0#202020)	0(0202)
12	2(1#20)	2(0#20)	3(31313131)	2(2020)
13	1(2020)	* 3(3131)	0(20#20202)	1(0202)
14	4(20##)	4(1###)	1(#3131313)	2(2020)
15	0(1202)	* 1(0313)	2(020#2020)	0(0202)
16	1(0313)	5(0###)	3(1#313131)	1(2020)
17	4(0313)	3(1#31)	0(2020#202)	0(0202)
18	2(1###)	* 2(2020)	1(31#31313)	2(2020)
19	1(##20)	2(0#20)	2(02020#20)	1(1313)
20	4(2020)	* 3(3131)	3(131#3131)	2(2020)
21	2(1###)	1(1313)	0(202020#2)	0(0202)
22	1(####)	1(0313)	1(3131#313)	1(1313)
23	0(0313)	0(0202)	2(0202020#)	0(0202)
24	2(1###)	3(1#31)	3(13131#31)	2(2020)
25	1(####)	1(2020)		1(1313)
26	4(0313)	2(0#20)		2(2020)
27	2(1#20)	0(#202)		
28	1(2020)	1(1313)		
29	4(0313)	1(0313)		
30	5(1###)	4(0###)		
31	1(#202)	4(1###)		
32	4(0313)	2(2020)		
33	2(1###)	6(0###)		
34	1(####)	4(####)		
35	4(2020)	1(1313)		
36	2(20##)	1(0313)		
37	1(#202)	0(0202)		
38	4(0313)	2(1#20)		
39	2(1313)	1(2020)		
40	3(1###)	3(0#31)		
41	4(0###)	0(#202)		
42	2(2031)	1(1313)		
43	3(2020)	1(0313)		
44	4(####)	3(3131)		
45	2(#313)	2(1#20)		
46	3(1###)	2(2020)		
47	4(0###)	3(0#31)		
48	2(#313)	4(####)		
49	3(1###)	4(1###)		
50	4(2020)	5(0###)		

heap size	octal *game* 0.36000		0.37000		0.45000		*0.53000*		0.54000	
0	*	0(1202)	*	0(1202)	*	0(1202)	*	0(1202)	*	0(1202)
1	*	1(0313)	*	1(0313)	*	0(1202)	*	1(0313)	*	1(0313)
2	*	0(1202)	*	2(2020)	*	1(0313)	*	1(0313)	*	0(1202)
3	*	2(2020)	*	0(1202)	*	1(0313)	*	2(2020)	*	1(0313)
4	*	1(0313)	*	1(0313)	*	2(2020)	*	2(2020)	*	2(2020)
5	*	0(1202)	*	2(2020)	*	2(2020)	*	1(0313)	*	2(2020)
6	*	2(2020)		3(1#31)	*	3(3131)		0(0202)	*	2(2020)
7	*	1(0313)	*	1(0313)	*	1(0313)	*	2(2020)	*	4(####)
8		3(1#31)	*	2(2020)	*	1(0313)		2(2020)	*	1(0313)
9	*	2(2020)		3(1#31)		4(1###)		4(####)		1(1313)
10	*	1(0313)		4(0###)	*	4(####)		0(0202)		1(#313)
11		3(1#31)		0(2020)	*	3(3131)		1(#313)	*	2(2020)
12	*	2(2020)		3(1#31)		2(2020)		2(2020)	*	2(2020)
13		4(0###)		4(0###)		2(2020)		2(2020)		2(2020)
14		3(1#31)		2(2020)		1(1313)		1(0313)		4(1###)
15		0(2020)		1(1313)		1(0313)		1(1313)		1(1313)
16		4(0###)		3(0#31)		4(0###)		2(2020)		1(1313)
17		3(1#31)		2(2020)		2(2020)		2(2020)		1(#313)
18		2(2020)		1(1313)		2(2020)		4(####)		2(2020)
19		4(0###)		0(0202)		6(####)		1(1313)		2(2020)
20		1(1313)		2(2020)		4(####)		1(#313)		2(2020)
21		2(2020)		1(1313)		4(####)		2(2020)		4(1###)
22		3(0#31)		4(0###)		1(1313)		2(2020)		1(1313)
23		1(1313)		5(####)		1(#313)		1(1313)		1(1313)
24		2(2020)		1(1313)		2(2020)		1(1313)		1(#313)
25		0(0202)		4(0###)		2(2020)		2(2020)		2(2020)
26		1(1313)		5(####)		7(####)		2(2020)		2(2020)
27		2(2020)		1(1313)		1(1###)		4(####)		2(2020)
28		4(0###)		2(0#20)		1(1313)		1(1313)		4(1###)
29		1(1313)		0(#202)		4(####)		1(1313)		1(1313)
30		5(####)		1(1313)		4(####)		2(2020)		1(1313)
31		4(0###)		2(2020)		3(3131)		2(2020)		1(#313)
32		1(1313)		3(3131)		2(2020)		1(1313)		2(2020)
33		5(####)		1(1313)		2(2020)		1(1313)		2(2020)
34		4(0###)		2(####)		1(0313)		2(2020)		2(2020)
35		1(1313)		3(3131)		1(1313)		2(2020)		4(1###)
36		5(####)		4(####)		4(####)		1(1313)		1(1313)
37		2(0#20)		2(#202)		8(####)		1(1313)		
38		1(1313)		3(31##)		2(2020)				
39		0(#202)		4(##31)		7(####)				
40		2(2020)		2(1###)		4(####)				
41		1(1313)		3(31#3)						
42		3(3131)		4(##31)						
43		2(####)		2(1#20)						

heap size		0.64000		0.72000		Octal *Game* 0.74000		*0.76000*		*0.77000*
0	*	0(1202)	*	0(1202)	*	0(1202)	*	0(1202)	*	0(1202)
1	*	0(1202)	*	1(0313)	*	1(0313)	*	1(0313)	*	1(0313)
2	*	1(0313)	*	0(1202)	*	0(1202)	*	0(1202)	*	2(2020)
3	*	2(2020)	*	2(2020)	*	1(0313)	*	2(2020)	*	3(3131)
4	*	3(3131)	*	3(3131)	*	2(2020)	*	3(3131)	*	1(0313)
5	*	4(####)	*	1(0313)	*	3(3131)	*	4(####)		4(1###)
6	*	1(0313)	*	0(1202)	*	2(2020)	*	1(0313)	*	3(3131)
7		5(1###)	*	2(2020)	*	4(####)		6(1###)		2(2020)
8		3(3131)	*	3(3131)	*	1(0313)	*	2(2020)		1(1313)
9	*	2(2020)		1(#313)		4(1###)	*	3(3131)		4(0###)
10		1(#313)		0(0202)		6(####)		4(####)		2(2020)
11		5(1###)		2(2020)	*	2(2020)		1(#313)		6(####)
12		4(####)		3(3131)	*	3(3131)		6(1###)		4(0###)
13		2(####)		1(#313)		2(2020)		7(####)		1(1313)
14		6(####)		0(0202)		1(1313)		3(3131)		2(2020)
15		8(####)		2(1#20)		5(####)		2(2020)		7(####)
16		1(#313)		3(2031)		1(1313)		1(#313)		1(1313)
17		2(1###)		1(#313)		7(####)		6(####)		4(####)
18		3(####)		0(0202)		6(####)		7(####)		3(3131)
19		7(####)		2(1#20)		8(2020)		5(####)		2(2020)
20		4(####)		3(2031)		5(####)		2(2020)		1(0313)
21		5(####)		1(#313)		1(####)		8(####)		4(####)
22		8(3131)		0(0202)		10(0313)		9(####)		6(####)
23		2(1313)		2(1#20)		11(1###)		6(####)		7(####)
24		9(2020)		3(2031)		2(2020)		5(####)		4(####)
25		5(####)		1(#313)		6(####)		2(2020)		1(#313)
26		4(####)		0(0202)		8(2020)		8(####)		2(2020)
27		7(####)		2(1#20)		4(####)		7(1###)		8(####)
28		6(1###)		3(2031)		5(####)		1(#313)		5(####)
29		8(####)		1(#313)		10(#313)		10(####)		4(####)
30		1(####)		0(0202)		6(####)		4(20##)		7(####)
31				2(1#20)		2(####)		3(3131)		
32				3(2031)		1(2020)		7(####)		
33				1(#313)		5(####)				
34				0(0202)		1(2020)				
35				2(1#20)		5(1313)				
36				3(2031)		6(####)				
37				1(#313)						
38				0(*0202*)						

heap
size Octal game 4.70000

```
0    * 0(12020202)
1    * 1(03131313)
2    * 2(20202020)
3    * 1(03131313)
4    * 2(20202020)
5    1(#3131313)
6    2(20202020)
7    1(#3131313)
8    2(1#202020)
9    1(20#31313)
10   2(131#2020)
11   1(2020#313)
12   2(13131#20)
13   1(202020#3)
14   2(1313131#)
```

APPENDIX II - GENERALIZED GENERA FOR MESSY GAMES

How to read Appendix II

Each column in the appendix is devoted to a different 2-digit octal game, and gives the blurry genus sequence of any linear combination of the heaps which are listed in that column. Each column is independent; this appendix makes no claims about sums of games in different columns.

For heaps which equal small nimheaps, the entry looks something like this:

2 *3

This indicates that a heap of **2** counters *is* equal to a nimheap of **3** counters.

For other heaps, the entry looks something like this:

h a, period p(s)

This means that we can replace the heap of h counters by the game called 'a', and use the pretense that $(p + s)\ a = s \cdot a$

We tabulate all linear combinations of the games $a, b,$ c, ... in each column, after the above pretenses have been considered.

For an example of how to use this table, suppose that we are playing Dawson's Kayles (octal code .07) and we are presented with the following position:

21 17 12 10 **7 4**

Look up .07 in Appendix II. The first part of the table tells us that the heaps of sizes 7 and **4** equal nimheaps of sizes 1 and **2** respectively.

Look up **12** and **17.** The relevant lines are

12 b, period 2(0)

17 b, period 2(0)

We see that we can pretend that $b + b = 0$, so we may ignore these two heaps.

The heap of size 10 is the game called 'a', the heap of size 21 is called 'e'.

We **look** up the line in the second part of the entry for ***.07*** which has a '1' under the headings 'a' and 'e', and find that the genus is (3131). Since the other two heaps (7 and **4)** add up to ***3,** we find that this sum is a $\mathcal{P}$ position.

In the table for .16, for the sake of brevity I have not printed the lines concerning **5,** 4 and 3 heaps of size 8 or 10 except where they disagree with the corresponding lines for 1,2 and 1 heaps respectively.

In the tables for **.45** and .77, I have observed that certain heaps behave like adders, and have consolidated the tables accordingly, noting which adders the heaps resemble.

.07 is called *Dawson's Kayles,* and was analyzed by him.

.77 called *Kayles,* was introduced by Dudeney [6] and Loyd [11].

Octal game

0.04000

```
 1 *0
 2 *0
 3 *0
 4 *1
 5 *1
 6 *1
 7 *2
 8 *2
 9 *0
10 *3
11 *3
12 *1
13 *1
14 *1
15 *0
16 a, period 2(0)
17 b, period 2(0)
18 *3
19 *3
20 c, period 2(0)
21 *2
22 *2
23 a, period 2(0)
24 a, period 2(0)
25 d, period 1(0)
```

dcba

```
0000 (1202)
0001 (1###)
0010 (1#31)
0011 (1###)
0100 (0#20)
0101 (0###)
0110 (0313)
0111 (0###)
```

0.06000

```
 1 *0
 2 *0
 3 *1
 4 *1
 5 *2
 6 *2
 7 *0
 8 *3
 9 *1
10 *1
11 *2
12 a, period 2(1)
13 b, period 2(1)
14 *3
15 c, period 2(0)
16 c, period 2(0)
```

cba

```
000 (1202)
001 (1#20)
002 (1202)
010 (1#31)
011 (1313)
012 (3131)
020 (1202)
021 (2020)
022 (0202)
100 (0###)
101 (0###)
102 (0###)
110 (0###)
111 (####)
112 (####)
120 (0###)
121 (####)
122 (####)
```

0.07000

```
 1 *0 (####)
 2 *1 (1313)
 3 *1 (2020)
 4 *2 (3131)
 5 *0 (0202)
 6 *3
 7 *1
 8 *1
 9 *0
10 a, period 2(0)
11 *3
12 b, period 2(0)
13 *2
14 c, period 2(0)
15 *0
16 d, period 2(0)
17 b, period 2(0)
18 *2
19 a, period 2(0)
20 *3
21 e, period 1(1)
```

edcba

```
00000 (1202)
00001 (1#31)
00010 (0#20)
00011 (0313)
00100 (1###)
00101 (1###)
00110 (0###)
00111 (0###)
01000 (0###)
01001 (0###)
01010 (1###)
01011 (1###)
01100 (0313)
01101 (0#20)
01110 (1#31)
01111 (1202)
10000 (0202)
10001 (3131)
10010 (2020)
10011 (1313)
10100 (####)
10101 (####)
10110 (####)
10111 (####)
11000 (####)
11001 (####)
11010 (####)
```

0.04000

0.06000

0.07000

11011 (####)
11100 (1313)
11101 (2020)
11110 (3131)
11111 (0202)

0.14000

```
 1 *1
 2 *0
 3 *0
 4 *1
 5 *0
 6 *2
 7 *1
 8 *2
 9 *2
10 *1
11 *0
12 a, period 2(2)
13 *1
14 b, period 2(0)
15 a, period 2(2)
16 c, period 2(2)

cba

000 (1202)
001 (####)
002 (0202)
003 (####)
010 (1###)
011 (2020)
012 (####)
013 (0202)
100 (3131)
101 (####)
102 (1313)
103 (####)
110 (####)
111 (1313)
112 (####)
113 (1313)
200 (0202)
201 (####)
202 (0202)
203 (####)
210 (####)
211 (0202)
212 (####)
213 (0202)
300 (1313)
301 (####)
302 (1313)
303 (####)
310 (####)
311 (1313)
312 (####)
313 (1313)
```

0.15000

```
 1 *1
 2 *1
 3 *0
 4 *1
 5 *1
 6 *2
 7 *2
 8 *1
 9 *2
10 *2
11 *1
12 *1
13 *0
14 a, period 2(2)
15 a, period 2(2)
16 *2
17 *2
18 b, period 2(2)
19 *2
20 c, period 2(0)

cba

000 (1202)
001 (#313)
002 (0202)
003 (1313)
010 (2020)
011 (3131)
012 (2020)
013 (3131)
020 (0202)
021 (1313)
022 (0202)
023 (1313)
030 (1313)
031 (0202)
032 (1313)
033 (0202)
100 (1#20)
101 (3131)
102 (2020)
103 (3131)
110 (0202)
111 (1313)
112 (0202)
113 (1313)
120 (2020)
121 (3131)
122 (2020)
123 (3131)
```

0.16000

```
 1 *1
 2 *0
 3 *0
 4 *1
 5 *2
 6 *2
 7 "1
 8 a, period 2(4)
 9 *0
10 *1
11 a, period 2(4)
12 b, period 2(0)
13 c, period 2(2)

cba

000 (1202)
001 (####)
002 (0202)
003 (####)
004 (0202)
005 (####)
010 (1#20)
011 (20##)
012 (##20)
013 (1###)
014 (2020)
015 (####)
100 (2020)
101 (####)
102 (1313)
110 (0202)
111 (####)
112 (3131)
200 (0202)
201 (####)
202 (0202)
210 (2020)
211 (####)
212 (2020)
300 (1313)
301 (####)
302 (1313)
310 (3131)
311 (####)
312 (3131)
```

Octal game

0.14000 0.15000 0.16000

130 (3131)
131 (2020)
132 (3131)
133 (2020)

```
0.17000

 1 *1
 2 *1
 3 *0
 4 *2
 5 *1
 6 *3
 7 "0
 8 *1
 9 *1
10 a, period 2(0)
11 *2
12 b, period 2(0)
13 *3
14 c, period 2(0)
15 *1
16 d, period 2(0)
17 a, period 2(0)
18 *2
19 b, period 2(0)
20 *3
21 e, period 2(1)

edcba

00000 (1202)
00001 (1#31)
00010 (0#20)
00011 (0313)
00100 (1###)
00101 (1###)
00110 (0###)
00111 (0###)
01000 (0###)
01001 (0###)
01010 (1###)
01011 (1###)
01100 (0313)
01101 (0#20)
01.110 (1#31)
01111 (1202)
10000 (1313)
10001 (2020)
10010 (3131)
10011 (0202)
10100 (####)
10101 (####)
10110 (####)
10111 (####)
11000 (####)
11001 (####)
11010 (####)
```

```
0.36000

 1 *1
 2 "0
 3 *2
 4 "1
 5 *0
 6 *2
 7 *1
 8 a, period 2(0)
 9 *2
10 *1
11 a, period 2(0)
12 *2
13 b, period 2(0)
14 a, period 2(0)
15 c, period 1(2)
16 b, period 2(0)

cba

000 (1202)
001 (1#31)
010 (0###)
011 (0###)
100 (2020)
101 (#313)
110 (####)
111 (####)
200 (0202)
201 (3131)
210 (####)
211 (####)
```

```
0.37000

 1 *1
 2 *2
 3 *0
 4 *1
 5 *2
 6 a, period 2(0)
 7 *1
 8 *2
 9 a, period 2(0)
10 b, period 2(0)
11 c, period 1(2)
12 a, period 2(0)
13 b, period 2(0)

cba

000 (1202)
001 (1#31)
010 (0###)
011 (0###)
100 (2020)
101 (#313)
110 (####)
111 (####)
200 (0202)
201 (3131)
210 (####)
211 (####)
```

0.17000 0.36000 0.37000

```
11011 (####)
11100 (0202)
11101 (3131)
11110 (2020)
11111 (1313)
20000 (0202)
20001 (3131)
20010 (2020)
20011 (1313)
20100 (####)
20101 (####)
20110 (####)
20111 (####)
21000 (####)
21001 (####)
21010 (####)
21011 (####)
21100 (1313)
21101 (2020)
21110 (3131)
21111 (0202)
```

Octal game

0.45000

```
 1 *0
 2 *1
 3 *1
 4 *2
 5 *2
 6 *3
 7 *1
 8 *1
 9 a, period 2(0)
10 b, period 2(2)
11 *3
12 2(1) like :2
13 2(1) like :2
14 2(1) like :5

ba

00 (1202)
01 (1###)
10 (####)
11 (2020)
20 (0202)
21 (####)
30 (####)
31 (0202)
```

0.64000

```
 1 *0
 2 *1
 3 *2
 4 *3
 5 a, period 2(2)
 6 *1
 7 b, period 2(0)
 8 c, period 2(1)

cba

000 (1202)
001 (####)
002 (0202)
003 (####)
010 (1###)
011 (2020)
012 (####)
013 (1313)
100 (3131)
101 (####)
102 (3131)
103 (####)
110 (####)
111 (1313)
112 (####)
113 (2020)
200 (0202)
201 (####)
202 (0202)
203 (####)
210 (####)
211 (2020)
212 (####)
213 (1313)
```

0.74000

```
 1 *1
 2 *0
 3 *1
 4 *2
 5 *3
 6 *2
 7 a, period 2(2)
 8 *1
 9 b, period 2(0)

ba

00 (1202)
01 (####)
02 (0202)
03 (####)
10 (1###)
11 (2020)
12 (####)
13 (0202)
```

0.76000

```
 1 *1
 2 *0
 3 *2
 4 *3
 5 a, period 2(1)
 6 *1
 7 b, period 2(0)
 8 *2
 9 *3
10 a, period 2(1)
```

ba

```
00 (1202)
01 (####)
02 (0202)
10 (1###)
11 (2020)
12 (####)
```

0.77000

```
 1 *1
 2 *2
 3 *3
 4 *1
 5 a, period 2(1)
 6 *3
 7 2(1) like :2
 8 2(1) like :5
 9 d, period 2(1)
```

da

```
00 (1202)
01 (1###)
02 (1202)
10 (0###)
11 (0202)
12 (####)
20 (1202)
21 (####)
22 (0202)
```

APPENDIX III

This system is an implementation of many of the computations necessary for the analysis of normal partizan games, as treated in Conway, On Numbers and Games. This system can be used interactively as a tabletop calculator for partizan games or as a set of utility functions for writing LISP programs to analyze whole families of positions.

The internal representation of a game is transparent to the user except for the observation that sums of games are stored as sums, rather than working out their options. For example, ↑+↑+* will be stored as ↑+↑+* rather than {0|↑}. This sort of conversion can be made by using the function UNSUM (q.v.).

In the list of functions which follows, I will use the words game, interval, and thermograph to indicate data types as follows:

game: The internal structure is transparent; the data type game corresponds to the concept 'game' as used in ONAG. A game can be printed by using PR (q.v.).

interval: Represented as a list of two pairs of real numbers. The first pair indicates the left (high) end of an interval of the real number line, the second the right (low) end. The second number of each pair is always either 1 or -1; +1 on the left (high) end of an interval indicates that the interval is closed on the left; -1 on the right (low) end indicates that it is closed on the right. So the interval ((1 1) (-1 1)) is the interval [1, 1) (writing high numbers to the left). An interval can be printed in this form by using DISPINT (q.v.).

thermograph: Represented as a list of (real-number, interval) pairs. The real-number of each pair indicates a temperature while the paired interval indicates the confusion interval of the thermograph cooled by that temperature. All corners of the thermograph are indicated. A thermograph can be drawn by using DRAW (q.v.).

Now follows a list of functions which are available (in addition to the usual LISP functions) for processing games. These will be presented in the following form:

NAME (arg1:type, arg2:type . . .):type <description>

where NAME is the name of the function being described, arg1, arg2, ... are names of the arguments to be passed, with their associated data types, and the last 'type' is the data type of the returned result (if any).

CLEQZ, CGEQZ, CEQZ (g:game):boolean

Returns T if appropriate relation holds between g and 0.

CEQ, CLEQ, CGEQ (g1:game, g2:game):boolean

Returns T if the appropriate relation holds between g1 and g2.

RIGHT (g: game): list of games

Returns list of right options of g.

LEFT (g:game): list of games

Returns list of left options of g.

MPLUS (g1:game, g2:game, . . .): game

Returns sum of g1+g2+... . gi may be real numbers as well as games, so (mplus 2 2) will return the game 4.

MDIFFERENCE (g1:game, g2:game):game

Returns g1-g2.

MG: (l1:game, l2:game, ... !| r1:game, r2:game . . .): game

Returns a game whose left options are l1, l2, etc. and whose right options are r1, r2, etc. Options may also be real numbers.

UNSUM (g:game):game

Equivalent to (MG (left g) !| (right g)), i.e., takes game g (represented as a sum) and converts it to a single summand. It is necessary to do this before canonicalizing a game if a canonical form of the entire sum is required (see canon).

CANON (g: game):game

Eliminates reversible and dominated moves from each summand of g.

CNUMBERP (g:game) :boolean

Returns T if g is a number, NIL otherwise.

CONFINT (g:game): interval

Return confusion interval of g. Also g is checked to see if it is a number. If so, its internal representation is changed to reflect this fact.

DISPINT (g:game)

Displays the confusion interval of game g.

PR (g:game)

Prints game g in legible form.

KNOW (g:game, name:atom)

Tells the system to print game g by the name NAME. In subsequent calls to PR any occurrence of game g will be printed as NAME. A long list of known games severely degrades the performance of PR. The list of known games can be reset with (SETQ gamelist NIL). Games which were printed with names will retain these names even after the known game list has been reset.

THERMOGRAPH (g:game) :thermograph

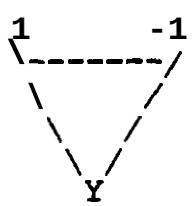
Returns the thermograph of game g. The last interval will be that of a number, which is the mean value of g.

DRAW (t:thermograph)

Draws a picture of thermograph t. Some of the information in t is not shown, since it doesn't fit on the diagram.

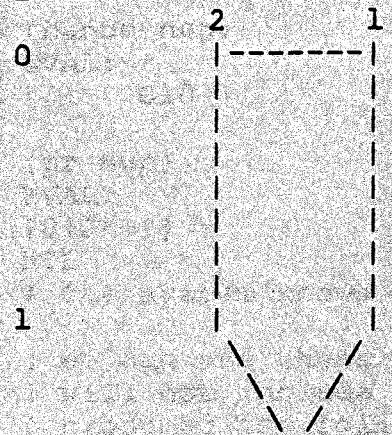
The following is a sample interactive session with the game system. Lines which begin with an ! are comments to explain the input and output.

```
! Use the LISP function setq and the game function mg to define
! the game plus-or-minus-one
(setq pml (mg 1 !| -1))
EVAL:
((1 . G6))
! LISP insists on printing out the internal representation.

! Use pr to print this game in a legible form
(pr pml)
EVAL:
{1|-1}
NIL

! Print its confusion interval
(dispint pml)
EVAL:
[1,-1]NIL

! and its thermograph
(draw (thermograph pml))
EVAL:
```

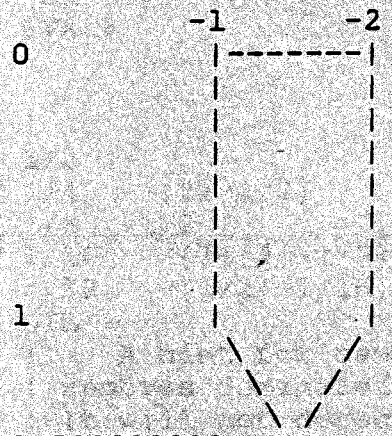



```
NIL

! Let's try a more complicated game
(setq a (mg (mplus 3 pml) (mplus 2 pml) !| pml))
EVAL:
((1 . G7))

(pr a)
EVAL:
{2+{1|-1},3+{1|-1}|{1|-1}}
NIL
```

```
(draw (thermograph a))
EVAL:
```




```
NIL

! How about the thermograph of its negative?
(draw (thermograph (mminus a)))
EVAL:
```




```
NIL

! Define the familiar game *={0|0}
(setq star (mg 0 !| 0))
EVAL:
((1 . G8))

! and up={0|*}
(setq up (mg 0 !| star))
EVAL:
((1 . G9))

(pr up)
EVAL:
{0|{0|0}}
NIL

! There must be a prettier way to print up
! So let's start by printing star as *
(know star '*)
EVAL:
((((1 . G8)) !*))
```

```
NIL

(canon nup)
EVAL:
((1 . G10))

(pr nup)
EVAL:
{0|2*+↑}
NIL
! The system knows that *-*=0, but not that *+*=0.

! We can use these functions to write programs, too. I
! will not include the source for the program which
! produced the following output, which is just a few
! lines of LISP code which compute values for the game
! 'baked alaska', which is played with integer arrays of
! squares. A move is for left to remove a vertical strip
! of squares (splitting the rectangle into one or two
! smaller rectangles).
(all 8)
EVAL:


1
1/2
1/4   -{2|-1/2}
1/8   1/2
1/16   -{1+{2|-1/2}|-1/8}   1
1/32   -{1/2|-1/16}   1/2
NIL
!    A hint for canonicalizing games; CANON simply
! removes dominated and reversible options from a game.
! It will not detect that a game is a number. However,
! when computing the confusion interval of a game, if it
! turns out to be a number, then the internal
! representation is replaced by a much simpler object.
! CANON is quite capable of dealing with this, and will
! perform better if all reductions of this kind have been
! done. Hence it is a good idea to ask for the confusion
! interval for a game before asking for its canonical
! form (this was done by the program which worked out
! baked alaska).
```

```
(pr up)
EVAL:
{0|*}
NIL

! Better, but up is a common game, so let's shorten it still more
(know up '↑)
EVAL:
((((1 . G9)) !↑) (((1 . G8)) !*))

(pr up)
EVAL:
↑
NIL

! It is expensive to check each new game for equality
! with up and star.
! so we  will not check any new games.
(setq gamelist nil)
EVAL:
NIL

! But the old games remember their names ...
(pr up)
EVAL:
↑
NIL

(setq dups (mplus up up star))
EVAL:
((2 . G9) (1 . G8))

! and tell them to new games which are defined in terms
! of the old ones.
(pr dups)
EVAL:
2↑+*
NIL

! Let's try to work out the canonical form for ↑+↑+*
(canon dups)
EVAL:
((1 . G8) (2 . G9))

(pr dups)
EVAL:
2↑+*
NIL

! But ↑ and * were already in simplest form.  So we work out the
! options of ↑+↑+*, so that we can canonicalize the entire sum
(setq nup (unsum dups))
EVAL:
((1 . G10))

(pr nup)
EVAL:
{*+↑,2↑|2*+↑,2↑}
```

1. Elwyn Berlekamp, John Conway and Richard Guy, Winning *Ways*, Academic Press, London and New York, 1982.
2. Charles R. Bouton, Nim, a game with a complete mathematical theory, Ann. of Math. Princeton (2), 3(1901-02) 35-39.
3. J. H. Conway, *On Numbers and Games*, Academic Press, London and New York, 1976.
4. T. R. Dawson, Fairy Chess Review (Dec. 1934) p. 94, problem 1603.
5. T.R. Dawson, Caissa's Wild Roses, 1935, p. 13.
6. H.E. Dudeney, Canterbury Puzzles, London, 1910, pp. 118, 220.
7. T.S. Ferguson, On sums of graph games with the last player losing, Internat. J. Game Theory, 3(1974) 159-167; M.R. 52#5046.
8. P.M. Grundy, Mathematics and Games, Eureka 2(1939) 6-8; reprinted *ibid.* 27(1964) 9-11.
9. P.M. Grundy and C.A.B. Smith, Disjunctive games with the last player losing, Proc. Cambridge Philos. Soc. 52(1956) 527-533; MR 18, 546.
10. Richard K. Guy and Cedric A. B. Smith, The G-values of various games, Proc. Cambridge Philos. Soc. 52(1956) 514-526; MR 18, 546.
11. Sam Loyd, Cyclopedia of Tricks and Puzzles, New York, 1914, p. 232.
12. R. P. Sprague, Über mathematische Kampfspiele, Tôhoku Math. J. 41(1935-6) 438-444; Zbl. 13, 290.
13. Yōhei Yamasaki, On Misère Nim-type games, J. Math. Soc. Japan, 32(1980) 461-475.