Games of No Chance
MSRI Publications
Volume **29**, 1996

The Gamesman's Toolkit

DAVID WOLFE

Abstract. Tired of all those hand calculations? Of converting to canonical forms? Of calculating means and temperatures? Of wondering if you've goofed? Wanted to play with the overheating operator, but don't have any patience?

The Gamesman's Toolkit, written in C under UNIX, implements virtually all of the material in finite combinatorial game theory. It is invaluable in analyzing games, and in generating and testing hypotheses. Several of the results presented at the 1994 Combinatorial Games Workshop were discovered using this program.

The Gamesman's Toolkit is useful and fun. This paper is an overview of its features.

1. Overview

The Gamesman's Toolkit, written in C and running under UNIX, implements virtually all of the material in finite combinatorial game theory. The toolkit can be used in one of two ways: as a "games calculator", by the gamesman who wishes to do standard algebraic manipulations on games; and as a programming toolkit, to analyze a particular game, for instance. A broad base of game-theoretic functions is provided, along with a parser and output routines. Either way, the program has proved to be a versatile and powerful tool for the student and the researcher.

2. Use As a Games Calculator

Table 1 shows a sample run of the Toolkit, and gives an idea of its functionality as a games calculator. Here is the input notation, in a nutshell:

- The caret, `^`, represents $\uparrow$; the letter `v` is reserved for $\downarrow$. Triple-up can be typed as either `^^^` or `^3`.
- You can assign variables to games and later reuse them. For example, you can set `g = 2|1` and later reuse `g` in another expression.

`unix% games`	The program is executed from the UNIX prompt.
`Type 'help' and...`	You are told how to get help; the prompt `>` appears.
`> * ? ^`	How does $*$ compare with $\uparrow$?
`<>`	Answer: they are incomparable
`> * ? ^^`	How does $*$ compare with $\Uparrow$?
`<`	Answer: $*$ is less than $\Uparrow$
`> -2 + 3/8 + vvv + *6`	Compute $-2+\frac{3}{8}+\downarrow\downarrow\downarrow+*6$
`-13/8v3*6`	Answer: $-\frac{13}{8}+\downarrow\downarrow\downarrow+*6$ (`v3` is short for `vvv`)
`> +[2],*|-2`	Find the canonical form of $\{+_2, *\mid -2\}$;
`0,*|-2`	brackets indicate subscripts, so `+[2]` is tiny-two
`> $[1/2]`	Cool the last result ($) by $\frac{1}{2}$
`-1/2|-3/2`	$\{0,*\mid -2\}$ cooled by $\frac{1}{2}$ is $\{-\frac{1}{2}\mid -\frac{3}{2}\}$
`> g = $`	Assign a variable g to the last result
`g = -1/2|-3/2`	
`> mean g`	Calculate g's mean and temperature
`-1`	
`> temp g`	
`1/2`	
`> (1|||0||-1|-3)[1/2]`	Cool $1\|\|\|0\|\|-1\|-3$ by $\frac{1}{2}$
`1/2||0|g`	The program uses the value of g to shorten the answer
`> no ups`	Please do not print out $\uparrow$ in canonical forms
`> ^*`	What's the canonical form of $\uparrow*$?
`0,*|0`	It's $\{0,*\mid 0\}$
`> domino`	Enter a domineering position
`Enter domin...`	
`xxxx`	
`  xx`	
`xxxx`	
	The input is ended with an extra carriage return
`-3/2|-4`	The ten-box position has value $\{-\frac{3}{2}\mid -4\}$

Table 1. Sample run of the Gamesman's Toolkit. The first column is the computer dialogue, and the second contains comments. What the computer prints is in typewriter font, and what the user types is underlined. Most commonly, the user enters a game expression, and the computer converts the expression to canonical form.

- Subscripts in games notation are typically enclosed by brackets for the toolkit. Thus, the input format for $+_2$ (tiny-2) is `+[2]`, and that for g_3 (i.e., g cooled by 3) is `g[3]`.
- Similarly, superscripts are enclosed by angle brackets `<` and `>`.
- The symbol `%` represents the heating (integral) symbol. Thus, `%[1*]<1>g` denotes $\int_{1*}^{1} g$, that is, g overheated from $1*$ to 1 [Berlekamp 1988; Berlekamp and Wolfe 1994, p. 214].

3. Use As a Programmer's Toolkit

The Gamesman's Toolkit can be extended to analyze a particular game: the user implements the rules to that game, the program can then evaluate positions of that game. If you are an experienced (or patient) programmer you should find the Toolkit quite versatile. (Unfortunately, the extensibility mechanism was not designed for users without programming experience.)

Several people have written such extensions. Konane (programmed by Michael Ernst), Toads and Frogs (Jeff Erickson) and Domineering (David Wolfe) come with the Toolkit. Dan Garcia has written a wonderful interface using Tcl for playing domineering on an X-window system [Garcia 1996].

To get you started, Table 2 summarizes the most useful files and functions you'll need. The listing on page 97 contains a program for evaluating positions in Wyt Queens. (Wyt Queens, or Wythoff's game [Berlekamp et al. 1982, p. 61], is played on a quarter-infinite chess board. One or more queens occupy a square each, and move independently. A move consists of moving a queen north, west, or northwest any number of squares. The last player to move wins.)

4. Availability

The Gamesman's toolkit is free, and is available by request: send e-mail to me at wolfe@cs.berkeley.edu. I've chosen to distribute it in this way, rather than by posting it publicly, so that I can maintain a mailing list for notifying you of significant improvements in the program. You can also download it from http://http.cs.berkeley.edu/~wolfe/games.tar.gz. Enjoy!

Acknowledgement

Elwyn Berlekamp first encouraged me to write the toolkit in a course at UC Berkeley in Spring of 1989. Many people have since contributed ideas and code to the program, including Dan Calistrate, Raymond Chen, Jeff Erickson, Michael Ernst, Dan Garcia, Yonghoan Kim, David Moews, and Martin Mueller.

gameops.h: Performs operations on games (game_type)	
init()	must be called once to initialize the game routines
make(list_type, list_type)	constructs a game from two lists of options and converts it to canonical form; destroys its arguments
zero	the zero game
num(Q_type), up(int), star(int)	construct games consisting numbers, ups and star
plus(game_type, game_type)	add two games
minus(game_type, game_type)	subtract two games
negative(game_type)	negate a game
eq(game_type, game_type), ge(...), etc.	compare games (equal, $\geq$, etc.)
is_int(game_type), is_num(game_type)	true if game is an integer, number
list_type left_options(game_type)	game's list of left options
list_type right_options(game_type)	game's right options
output.h: Output routines for games and lists	
game_printf(game_type)	output a game plus a newline
game_print(game_type)	output a game without the newline
game_sprintn(game_type)	print a game to a string without newline, maximum of n characters
list.h: Manipulates lists of integers or games (list_type)	
list_make()	returns an empty list
list_insert(list_type, int or game_type)	insert an element into a sorted list
list_prepend(list_type, int or game_type)	prepend to an unsorted list
int or game_type list_nth(list_type)	find the nth element of a list
list_copy(list_type)	return a copy of a list
list_free(list_type)	destroy a list and free its space
rational.h: Manipulates low precision rational numbers (Q_type)	
Q(int, int)	construct a rational p/q
int Q_p(Q_type), Q_q(Q_type)	numerator or denominator of a rational
hash.h: Maintains a hash table keyed by an (int, list_type) pair	
boolean hash_test(int, list_type)	true if the hash table contains entry
game_type hash_get_last()	get the last entry tested positive
hash_put(int, list_type, game_type)	put a value into the hash table; destroys its list argument

Table 2. Most common functions used by a programmer of the toolkit.

```
/* A programming example: The game of Wyt Queens */
#include "games.c"       /* Includes all the needed .h files */
#define QUEENS_KEY 1001  /* Hash table keys below 1000 are reserved. */

game_type queens (int x, int y) {
    list_type posn_as_list, left, right;
    game_type g;
    int n, min;

    /* Encode the position as a list of integers, in order to use the hash
       table to store computed positions and avoid recomputing positions */
    posn_as_list = list_make();
    list_prepend (posn_as_list, x);
    list_prepend (posn_as_list, y);
    if (hash_test (QUEENS_KEY, posn_as_list)) { /* If pos'n previously computed */
        list_free (posn_as_list);               /* Free space used by the list */
        return hash_get_last();                 /* Return computed value */
    }

    /* Position wasn't already computed, so evaluate the position recursively. */
    left = list_make();
    for (n=0; n<x; n++) list_insert (left, queens (n, y)); /* Horizontal moves */
    for (n=0; n<y; n++) list_insert (left, queens (x, n)); /* Vertical */
    min = ( x<y ? x : y);
    for (n=1; n<=min; n++) list_insert (left, queens (x-n,y-n)); /* Diagonal */
    right = list_copy (left);       /* Right's options are the same as Left's. */
    g = make (left, right);         /* Construct the game's canonical form.
                                       Lists left and right are destroyed. */

    /* Store the value of the position in hash table.  The posn_as_list is
       destroyed and freed by hash_put() by, so no need to list_free() it. */
    hash_put (QUEENS_KEY, posn_as_list, g);

    return g;
}

#define MAXBUFF 10
void main() {
    char s[MAX_BUFF];

    init();                                  /* initialize the toolkit! */
    game_sprintn (s, queens(2,6), MAXBUFF-1); /* Store position in string s */
    printf ("A queen at location (2,6) has value %s\n", s);
}
```

Listing 1. Sample extension program for the Toolkit. Lines marked with a vertical bar on the margin relate to the hash table, and may be ignored on a first reading, or omitted if efficiency is not an issue. The hash table is used to avoid the exponential cost of reevaluating the same positions over and over during recursion.

References

[Berlekamp 1988] E. R. Berlekamp, “Blockbusting and Domineering”, *J. Combin. Theory* (Ser. A) **49** (1988), 67–116.

[Berlekamp and Wolfe 1994] E. Berlekamp and D. Wolfe, *Mathematical Go: Chilling Gets the Last Point*, A. K. Peters, Wellesley, MA, 1994.

[Berlekamp et al. 1982] E. R. Berlekamp, J. H. Conway and R. K. Guy, *Winning Ways for Your Mathematical Plays*, Academic Press, London, 1982.

[Garcia 1996] D. Garcia, “Xdom: A graphical, X-based front-end for Domineering”, pp. 311–313 in this volume.

David Wolfe
Computer Science Division
UC Berkeley, CA 94720
wolfe@cs.berkeley.edu