

# Intervalo and Nomignolov

Jean M. Morales  
JeanM\_M@yahoo.com  
nomignolov.blogspot.com

Oct 2007

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Intervalo's rules</b>	<b>2</b>
<b>3</b>	<b>Algorithms</b>	<b>2</b>
<b>4</b>	<b>Nomignolov's implementation</b>	<b>3</b>
<b>5</b>	<b>Results and comments</b>	<b>5</b>
<b>6</b>	<b>Final words</b>	<b>7</b>
<b>A</b>	<b>Tournament's results</b>	<b>7</b>

## 1 Introduction

Here's how *Nomignolov* version 1.2 "Benbow" was used to rank the strenghts of algorithms that play to *Intervalo*.

Nomignolov is a free universal program that provides a graphics interface to play at abstract strategy games, i.e. games where there's no hidden information and where there's no random elements other than the players' opinions about which move is their best. You can learn more about Nomignolov, and download it, at [nomignolov.blogspot.com](http://nomignolov.blogspot.com).

Intervalo is a game by Marcos Donnantuoni, derived from his game *Frame* of 2003. I'll describe the game's rules, which are very simple. This game is not "abstract" in the sense described above, because the two players moves simultaneously. It has hidden information.

Marcos, from his site [bitsenelring.blogspot.com](http://bitsenelring.blogspot.com), accepts algorithms that analyse the situation of the game and make a move. He makes them play

at Intervalo in a digital tournament. How to know if my algorithm has a chance?

Even if Nomignolov is developed with abstract strategy games in mind, it's possible to implement any sort of games <sup>1</sup>. I implemented Intervalo, and selected my best algorithm to compete to the tournament, and here's the full story!

## 2 Intervalo's rules

The board of the game is a line of 64 cells, named 1 to 64, initially empty.

In each turn, players think of a move, which means they think of an empty place to put one of their pieces. Once both have chosen, they say together the move (like rock-scissors-paper).

There is a small probability (which grows as the game proceeds) that the selected cell is the same. In this case a neutral piece is put in the cell

If the cells are different, each player occupies the announced cell with one of his pieces. The two pieces comprise an *interval* of cells. The player with the majority of pieces in that interval gains one point. Neutral pieces doesn't count.

At the end of the game, which, for a very boring one, consists of at most 64 turns, the player with most points wins the game.

## 3 Algorithms

Thinking of Intervalo, my sister Marisa and I found a total of 11 algorithms.

**1 - Oscillator** The first cell considered is 64 if the algorithm has more pieces in  $[33, \dots, 64]$  than in  $[1, \dots, 32]$ , otherwise the first cell is 1. Starting from the first cell considered, inspects all the cells and chooses the first cell found empty.

**2 - Halver** Finds the longest run of empty cells, taking the first one found in case of a draw. Chooses the cell in the middle of this sequence, taking the middle-right cell if the sequence has an even number of cells.

**3 - Filler** Chooses the first empty cell, starting from cell 1.

**4 - Center and Borders (Marisa)** Deduces which turn is it from the number of pieces on the board. For the first 4 turns, the algorithm plays with the strategies *CR*, *BL*, *CL*, *BR*, then repeats. Each strategy finds the first empty cell: *CR* in  $[33, \dots, 64]$ ; *BR* in  $[1, \dots, 32]$ ; *CL* in  $[32, \dots, 1]$ ; *BL* in  $[64, \dots, 33]$ . If half of the board is complete, it oscillates in the other half only.

**5 - Median (Marisa)** Counts the number of empty cells, and chooses cells  $x$  if in  $[1, \dots, x - 1]$  there are the same number of empty cells as in

---

<sup>1</sup>Nomignolov "knows" about a game with a Lua script — for Lua language information visit [www.lua.org](http://www.lua.org)

$[x + 1, \dots, 64]$ . Leaves one more empty cells in the left set if the number of empty cells is even.

**6 - Random** Chooses one empty cells at random.

**7 - Bodyguard** Finds the first opponent's piece with an empty cell to the left or to the right. If not such a cell exists, moves at random.

**8 - Inner Oscillator (Marisa)** Chooses the first empty cells with the order  $32 + 1, 33 - 1, 32 + 2, 33 - 2, 32 + 3, 33 - 3, \dots$

**9 - Primer (Marisa)** Chooses the first empty cell looking first at cells with prime numbers  $(2, 3, 5, \dots, 61)$ . If they are all occupied, plays at random.

**10 - Mirror** Chooses the first empty cell  $i$  from  $[1, \dots, 32]$ , where  $65 - i$  is occupied. Else plays at random.

**11 - Even** Chooses the first empty cell  $i$  when in  $[1, \dots, i - 1]$  there are more friendly pieces than opponent's pieces (but at least 1 piece per player).

Some of these algorithms are strong, some are very weak. Some can't win even in theory.

## 4 Nomignolov's implementation

Once I had all these algorithms, I thought how Nomignolov could help me finding the strongest. I faced three different hindrances. There's how I overcame them.

**How to move simultaneously?** This is tricky. Nomignolov was developed having one player per turn in mind. In each turn, it lists all the possible moves for the current player, and that can't be changed.

I got around this in the

```
make_move
```

function. The "move" for the first player is saved in a global flag, like this:

```
set_flag(name_to_id(move));
```

but has no effect on the board. Then it's the second player's turn. In the Listbox, the move of the first player is displayed, so if you don't want the temptation of looking at the AI move, let the program be the first player. This is a big problem if two people wished to play, but it's no a big concern of mine...

When the function is called for the second player, the flag is read and the move is completed at once.

**Where and how to define a move-driven algorithm?** This is tricky. Nomignolov implements a simple minimax algorithm. In a word, in order to choose one move, Nomignolov lists all the possible moves, generates the situations of the game after each move, and evaluates the game's situations, picking the move with leads to the best game's situation.

The algorithms in §3 follow a logic that's completely different. They describe how to choose a move, without addressing the problem of evaluating and of ordering the game's situations.

I wrote the 11 algorithm as functions:

```
ALG_01 = function (situation)
```

that return the id of the selected move. The parameter "situation" is a string with 64 chars, with ".", "n", "p", "a" for an empty cells, a neutral piece, a piece of the algorithm, a piece of the opponent).

I didn't want to change the logic of minimax, so I let Nomignolov play "at random". Then I modified the function

```
list_moves
```

so that there's just one possible move to choose from.

When I wanted algorithm  $x$  to play with algorithm  $y$ , if the game is not finished and the player is the first (id\_alg\_A is the id of algorithm  $x$ ), the list move is simply

```
move = cells[ALG(id_alg_A, s_situation)];  
n_move = 1;  
moves[n_move] = move;
```

**How to sort algorithms by strength?** This is tricky. The problem is that version "Benbow" of Nomignolov has no way to repeat a game many times. Having access to the source code, I solve this quite easily. . .

When Nomignolov finds that the game ended, in my customized version I made it print a line in "PlayedGames.txt" with the format

```
progressive_number: name_player_1 vs name_player_2 outcome
```

where "outcome" is  $-1$ ,  $1$ ,  $2$  in case of a draw, wins the first player, wins the second player.

In the function

```
init_graphics
```

of "Intervalo.lua", I read the file "PlayedGames.txt". If each algorithm challenges all the others  $r = 100$  times, and since there are  $n$  algorithms, a total of  $n \frac{(n-1)}{2} r = 5500$  games are played <sup>2</sup>. The function deduces which algorithms play in the current match, and so writes the variables id\_alg\_A and id\_alg\_B used by the list\_moves function.

The tournament took some 4 or 5 hours. There are many bottlenecks: the repeated I/O operations and the update of the graphics (quite useless

---

<sup>2</sup>The proof is too simple and will be omitted. . . Notice however that there is not first player advantage, so it's enough to have  $r$  matches  $x$  vs  $y$ , because matches  $y$  vs  $x$  are the same.

here) are the main. One advantage of keeping the game number in a *txt* is that the tournament can be interrupted and restarted.

Having solved all the problems, with a simple Lua script, and some R code, I read eagerly the outcome of the tournament!

## 5 Results and comments

In Table 1, the column with points shows the number of game won by the algorithm. In a total of 5500 games, 4976 games ended with the victory of one algorithm, an 524 were draws.

position	algorithm	name	points
1	5	Median	748
2	2	Halver	670
3	10	Mirror	610
4	8	Inner Oscillator	601
5	6	Random	599
6	11	Even	578
7	7	Bodyguard	528
8	9	Primer	364
9	4	Center and Borders	278
10	1	Oscillator	0
11	3	Filler	0
		Total	4976

Table 1: Results of domestic tournament

Noting that each algorithm played a total of exactly  $r(n - 1) = 1000$  games, the percentage of victory is self-evident, and I didn't report it.

The Oscillator and the Filler cannot win. This is because they fill the board starting from the border, and cannot enclose pieces of their own this way.

It's interesting to notice that the exact opposite of the loser algorithms, namely the Inner Oscillator, is not enough to have a winning algorithm.

The winner algorithm, Median, is a variation of Inner Oscillator. It moves near the center of the board, but takes into account the opponent's moves. More moves in the left side of the board, means that Median will play to the right, and vice versa. Evidently this helps to balance the opponent's game end encloses the maximum of its own pieces.

Another interesting thing to notice is the position of the Random algorithm: it is very close to position 4, and given it's nature, it could have been in a slightly better (or worse) position.

In Tables 2 and 3 I list details of the points shown in Table 1 and of the number of victories collected by each algorithm.

	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$	$a_{11}$
$a_1$	0	0	0	0	0	0	0	0	0	0	0
$a_2$	100	0	100	100	0	63	26	100	100	38	43
$a_3$	0	0	0	0	0	0	0	0	0	0	0
$a_4$	100	0	100	0	0	1	0	0	51	0	26
$a_5$	0	100	100	100	0	77	100	0	100	100	71
$a_6$	100	37	100	98	19	0	36	22	90	51	46
$a_7$	98	61	100	100	0	52	0	0	0	61	56
$a_8$	100	0	100	0	0	72	100	0	86	76	67
$a_9$	100	0	100	35	0	7	100	8	0	9	5
$a_{10}$	100	58	95	100	0	45	35	21	89	0	67
$a_{11}$	100	49	100	72	19	49	40	25	95	29	0

Table 2: Details of  $x$  vs  $y$  matches. The value in row  $i$ , column  $j$  is the number of victories of  $a_i$  vs  $a_j$ .

symbol	name	points	victories
$a_5$	Median	748	8
$a_2$	Halver	670	6
$a_{10}$	Mirror	610	6
$a_8$	Inner Oscillator	601	7
$a_6$	Random	599	5
$a_{11}$	Even	578	4
$a_7$	Bodyguard	528	7
$a_9$	Primer	364	3
$a_4$	Center and Borders	278	3
$a_1$	Oscillator	0	0
$a_3$	Filler	0	0
	Total	4976	49

Table 3: Number of victories per algorithm

In Table 2, algorithm  $i$  was named  $a_i$  for brevity. This table is useful to show the relations between algorithms.

Table 3 doesn't change the winner listed in Table 1, but shows that the number of points is not ordered as the number of victories! This is because algorithm  $x$  beats algorithm  $y$  if it wins *more* than the half of the time ( $\frac{r}{2} = 50$ ).

For instance, the Inner Oscillator  $a_8$  beat one more algorithm than

the Halver  $a_2$ , but with a lower margin. A similar remark may be done for Bodyguard  $a_7$ . It beat 7 algorithms, but the margin was so low that this was probably by chance.

One last interesting thing is evident from the Table 4: algorithms  $a_{10}$  and  $a_{11}$  are beaten by  $a_5$ , but beat  $a_2$ .

	$a_2$	$a_5$	$a_{10}$	$a_{11}$
$a_2$	0	0	38	43
$a_5$	100	0	100	71
$a_{10}$	58	0	0	67
$a_{11}$	49	19	29	0

Table 4: Results of algorithms  $a_2$ ,  $a_5$ ,  $a_{10}$  and  $a_{11}$

The effect of this order is that, when I first run the tournament prior to the inclusion of the last two algorithms,  $a_2$  was the winner (so my “best” algorithm). But afterward,  $a_5$  became the “best” winning 2 times more. This means that the outcome of the tournament depends on the challengers!

Also, the presence of randomic algorithms leads to the questions whether  $r = 100$  games are enough, or if winning 51 times over 100 is a sufficient margin ( $a_6$  beat  $a_{10}$  exactly 51 times ...)

## 6 Final words

That was fun! Thanks to Marcos for the idea.

Yes, the submitted algorithm,  $a_5$ , was from my sisters Marisa, and yes, it beat  $a_2$  after I added  $a_{10}$  and  $a_{11}$  :-)

And no, I don’t think I’ll win the tournament...I enjoyed a lot coding Intervalo for Nomignolov and making my domestic tournament, that I fear the actual time spent thinking about “intelligent” algorithms was a bit neglected...

Go on and experiment with Nomignolov! You can download the “Intervalo.lua” and “Intervalo\_algoritmos.lua” (with the functions) files from the blog.

Bye!

## A Tournament’s results

In the official tournament there were 16 entries, and each algorithm played against each other algorithm 300 times (a total of 4500 games per algorithm).

Algorithm  $a_5$  came 8th, gaining a total of 2005 games. The Spanish name of this algorithm is “Mediana”. Algorithm  $a_2$ , whose Spanish name

is "Midadero", came 15th and gained a total of 854 games. What a shame!  
:-)