

Chess pieces and Nomignolov

Jean M. Morales
JeanM_M@yahoo.com
<http://nomignolov.blogspot.com>

Feb 2008

Contents

1	Intro	1
2	Move's internal structure	1
3	List_moves for chess-like games	3
4	Rewriting Tablut's list_moves	4
5	Outro	6

1 Intro

A lot of abstract strategy games, or board games in general, involve the displacement of pieces over the board. The game of Chess is a familiar example.

Nomignolov is designed for the widest range of games. This means that it is not bundled with facilities for chess-like games. Until now! I present here a general structure for listing all the moves in a chess-like game. Keep reading!

2 Move's internal structure

Nomignolov is a general program instead of a program specialized in just one game or one type of games. For this reason, it is impossible to have an internal format for a move optimized in any way.

The representation of a move is tightly related to the representation of the cells that form the board. But, as a game could have any type of board formed by n cells (each cell has a unique identifying number $1, \dots, n$), no standard and meaningful way of naming these cells could exist. The labelling

convention for a chessboard is very convenient for games on a squared board: the bottom-left square is $a1$, the bottom-right square is $h1$ and so on, having a letter of each column and a number for each line.

The great majority of games has either a regular board (convenient labelling conventions exist for hexagonal and triangular boards, etc) or a board simply assimilable to a regular one (the star in the Chinese Checkers, a rectangle with holes, etc). For totally irregular boards, naming each cell is still possible, and I can think of no obvious reason why not to name a cell. Consequently, the solution I adopted for a move representation, that usually deals with cells and cells' names, was the use of strings.

This has of course a negative effect on efficiency, as working with strings takes in general more time than to work with integers or vectors.

The main positive effect is that the move is intelligible by a human, once the move convention is known. At the time that this is written, I have a list of 19 scripts for Nomignolov ¹, so I can make some examples of moves.

A great many games have moves of just two types:

- insertion of a piece on the board (or removal of a piece from the board);
- displacement of a piece over the board (chess-like games).

The moves that involve pieces entering or exiting the boards are just movements of pieces from or to invisible cells. This conserve the number of pieces during the game.

Move Out → **Board**. In many games you just drop a piece in one cell. In this case the move is just the name of the destination cell, as $e4$.

Scripts that use this type of moves are: Tic-tac-toe, Hex, Sicic, Intervalo, Knight's tour, Sliding block puzzle editor, Dominono.

Move Board → **Out**. I have just one script where the active move is the removal of one piece from the board: the editor of Sliding block puzzle (when the proper tool is selected). Again, the name of the move is just the name of the cell.

Move Board → **Board**. The move has the format *from-to*, like $b2-c1$.

Scripts with this type of moves: Conway's Soldiers, Dao, Entropy, 8 Queens, Tablut, Magic square (4×4), Cibilimny.

Some games uses moves with the same format as *cell*, but with different meaning. For instance, the format *cell* is used in Wari and in Diffusion to select the cell from which the seeds are taken; in 14-15 puzzle, the tile to move; in FootSteps the bid.

Special moves. Many games have moves completely different from the two types seen here. This justify the use of the string format.

Scripts with special moves: Sliding block puzzles, where a move is nd (n is the identifying number of the moving block, and d is the direction of

¹These are the good old days when Nomignolov is still Version 1.2 "Benbow"!

the movement); Wari has the move *PASS* when you have no more pieces on your side, and Entropy and Tablut have the same move when no moves are available; 3verse, where a move is like *a1-a5-e5-e4*.

3 List_moves for chess-like games

For chess-like games, that is games with pieces on board where in general the move is the displacement of one piece, and has the format *from-to*, the list of all the moves has a fixed structure. Given the general setting where there are more types of pieces, each with a different movement, that structure can be as follows:

1. list each type of pieces;
2. list all pieces for each type;
3. list all movements for each piece according to its type;
4. add the found moves to the list.

To allow a simple structure of this type, some functions are needed. The function `get_cells_type(type)` takes the type of a given piece (for instance *swedes* in Tablut) and returns the list of cells where there is at least one piece of this type.

Since this function returns a vector, it should be called between curly brackets. The code

```
cells_type = {get_cells_type(pieces[1])};
```

puts in the vector `cells_type` the result of the function.

The list of all possible moves for a piece of a given type depends on the type that the game requires. Chinese Chess move in a different way from Western Chess. Such a list for a rook (which shares the movements with pieces in Tablut) is the output of the function `rook_moves` in the file “utility_square.lua”.

The function `rook_moves` takes the name of a cell (where a rook is supposed to be) and returns the list of the cells reachable by that rook.

To increase software reusability, all movements of Chess pieces should be handled by functions to be saved in the same file ². Different functions will be needed if one wished to write a chess-like game on an hexagonal board.

Note the difference between the two functions mentioned here: the first function, `get_cells_type()`, is an API function, meaning that it is part of the code of Nomignolov; the second function, `rook_moves`, is a custom function that lies in a middle layer between Nomignolov and the scripts.

²If you write functions for more pieces, send them to me and I'll include them in the next Nomignolov release!

4 Rewriting Tablut's `list_moves`

Before. Here is the function `list_moves` of Tablut *before* the use of the functions and structure described early. The first part contains some definitions of local variables ³:

```
list_moves =
  function ()

    local move = ""; local n_moves = 0;
    moves = {};

    local player = id_whose_turn_is_it();
    local piece;

    local d = {"n", "s", "e", "w"};
    local empty_cells = {};
```

In the second part of the function it looks for each piece and adds every possible moves in the four directions. Just types 1 and 3 (“swedes” and “swedish king”) are movable by player 1, and just type 2 (“muscovites”) is movable by player 2. Here is the code:

```
    for id = 1,n_cols*n_rows do
      piece = id_get_pieces(id);

      if ((player == 1) and
          ((piece == 1) or (piece == 3))) or
          ((player == 2) and (piece == 2))) then
        for i = 1,4 do
          empty_cells = seen_cells(id, d[i]);
          for k,v in ipairs(empty_cells) do
            move = cells[id] .. "-" .. cells[v];
            n_moves = n_moves + 1;
            moves[n_moves] = move;
          end
        end
      end
    end
  end
```

In the last part, if no moves were found, the *PASS* move is added:

```
    if (n_moves == 0) then
      n_moves = 1;
```

³All variables, even if defined inside a function, are global in Lua. The only variable that needs to be global in this function is `moves`, with the generated list of moves.

```

        moves[n_moves] = "PASS";
    end

    return n_moves;

end

```

After. The first part of the function, with the definitions, is modified as follow:

```

list_moves =
function ()

    local n_moves = 0; moves = {};
    local player = id_whose_turn_is_it();
    local player_pieces = {};
    local piece_moves = {};

    if (player == 1) then
        player_pieces[1] = "swedes";
        piece_moves[1] = rook_moves;
        player_pieces[2] = "swedish_king";
        piece_moves[2] = rook_moves;
    else
        player_pieces[1] = "muscovites";
        piece_moves[1] = rook_moves;
    end
end

```

I put in the vector **player_pieces** all the type of pieces belonging to the current player. This is not a big deal for Tablut, but in a game like Chess, where each player can move 6 different pieces, this is very convenient.

The vector **piece_moves** contains all the *functions* used by all the types of pieces. This is superfluous in Tablut, since I need just the function **rook_moves**, but I showed it here for demonstration.

The second part of the function is:

```

    for j,w in ipairs(player_pieces) do
        for k,v in ipairs({get_cells_type(w)}) do
            if (v ~= "captured") then
                for kk,vv in ipairs(piece_moves[j](v)) do
                    n_moves = n_moves + 1;
                    moves[n_moves] = v .. "-" .. vv;
                end
            end
        end
    end
end
end
end

```

In this part I list each type of pieces in vector **player_pieces**, then I list all the pieces for each type (stored in vector **{get_cells_type(w)}**), and finally add all the movements for that piece. I use the function for that type of pieces, **piece_moves**. This is precisely the structure I described in Section 3.

Note that I can't move a piece from the "captured" cell, that is, the off-board cell where all captured pieces are moved. I have to filter it out because the function **{get_cells_type(w)}**, not knowing the rules of the game, returns also the pieces in that cell.

The last part of the function is the same as before.

The length of the function is nearly the same as before, but it's structure is far more general and could be reused for more complex games.

5 Outro

Note that what I showed here is not "code development" for Nomignolov, but it is just the "thickening of the script layer" ⁴ :-)

Note too that I didn't feel like drawing a Chess table anywhere, and this is quite remarkable :-)

This issue was very short, because I have to work on the code! Version 1.3 of Nomignolov is coming!

⁴Actually, the function **{get_cells_type(w)}** wasn't there before this article was written. . . Anyway this is not the point! The point is that something like that could be written by a Nomignolov user. You, for instance!