

# Sliding block puzzles and Nomignolov

Jean M. Morales  
JeanM\_M@yahoo.com  
<http://nomignolov.blogspot.com>

Dec 2007

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Abstract games</b>	<b>1</b>
2.1	A personal story . . . . .	1
2.2	The class of abstract games . . . . .	2
<b>3</b>	<b>What's in a script</b>	<b>3</b>
3.1	Sliding block puzzles . . . . .	3
3.2	Editor script . . . . .	5
3.3	Comments . . . . .	6
<b>4</b>	<b>Conclusions</b>	<b>6</b>

## 1 Introduction

You keep reading everywhere (and let me tell you you shall be very proud of your readings. . .) that Nomignolov handles just abstract strategy games. That's not the complete true, as I'll explain at length in this article.

I'll apply Nomignolov to Sliding block puzzles, showing how to handle differently shaped pieces as macro-pieces. A general approach of saving the pieces' configuration to a file leads directly to the concept of variants.

In the spirit of parting even more (but just for this article!) from the abstract game, I present a script that let's you "play" at an *editor* for Sliding block puzzles!

## 2 Abstract games

### 2.1 A personal story

Nomignolov borns from my love for abstract games.

I certainly know how to play the Italian version of checkers since I was a child. At the secondary school I used to play Gomoku (5 in a row) on my notebooks, and that was one reason more to like maths and paper with lots of little squares.

Of course I knew the rules of chess, even if knowing how to play is another thing. But how can't one help being fascinated by the existence of countless fairy pieces and variants?

And I knew and have even played a few or a hundred games at Mancala (which I now call Wari), Connect Four, Nine Men's Morris, Abalone, Othello, etc.

Finally, one day, I discovered that all these games, and literally hundreds more I wasn't aware of, are *abstract* games! I thank Luca Cerrato for his most interesting aperiodic Italian e-zine, "Il Fogliaccio degli Astratti" <sup>1</sup>, where he presents a lot of games. He also introduced me to Richard's play-by-e-mail server <sup>2</sup>, where you can play for free to lots of games with people from every corner of the world.

In a few months I played hundreds of games and learned the rules of tens more. The problem is that often, by just reading the rules, one can't get the gist of a game. Well: that's exactly <sup>3</sup> what I wrote Nomignolov for!

## 2.2 The class of abstract games

I call *abstract* a game with no random elements or hidden information, although someone argues that another trademark of abstract game is the themelessness. I simply disagree.

Figure 1 shows the relationships between games in general, abstract games, puzzles and cellular automata.

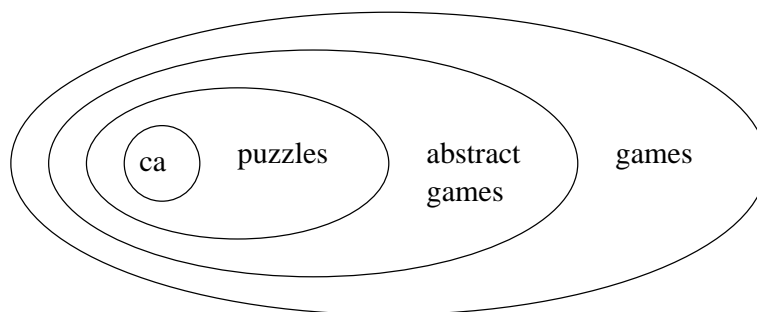


Figure 1: Abstract games as a subset of games and superset of puzzles

---

<sup>1</sup><http://www.tavolando.net/>;The title means something like "the bad paper of abstracts".

<sup>2</sup><http://www.gamerz.net/~pbmserv/>; My account there is "jeanm\_m". You are welcome to challenge me there at any game at any time, but I can't assure you to accept. . .

<sup>3</sup>Anyhow, that's the main reason. . .

A game in general has a number of random elements. The random elements can be thought as the actions of a random player (the *Nature*). Abstract games are games without the random player. Puzzles are abstract games where there is only one player. Cellular automata are puzzles where the only player has just one move available per turn.

### 3 What's in a script

Nomignolov handles virtually every game (and so abstract, puzzle or cellular automata). And more. To clarify this point, I need to say that a script provided to Nomignolov describes an entity with a logical part and a material part.

The logical part consists of the rules of the game. In particular it lists all possible moves (the possible actions), and it must determine if a player won the game (the final situation).

The materials are a certain number of pieces, of a certain number of different kinds, that can be moved in a certain number of positions, the *cells*.

This setting leaves to the programmer the greatest flexibility. I describe in Section 3.1 how to write a script to play at a Sliding block puzzle, and in Section 3.2 how to write a script to have a simple *editor* for these puzzles!

#### 3.1 Sliding block puzzles

In a Sliding block puzzle there is a number of macro-pieces (blocks) which may slide on a board and try to reach a given final position. A macro-piece is a sole piece formed by a set of pieces from different cells, whose relative position is fixed. Since I focus here on squared boards with squared cells, the blocks are just *polyminoes* and blocks may move only orthogonally.

Having to implement Sliding block puzzles in Nomignolov, I consider a macro-piece formed by all pieces on board of the same type. As a result, a block could be a *pseudo-polyminoes*, that is formed by squares that may not touch. Examples of polyminoes and pseudo-polyminoes are showed in Figure 2.

The final position may requires to correctly place a single block regardless of the position of the others. This objective may be imposed to more or all the blocks.

A famous example of this second type of puzzles is the 14-15 puzzle by Sam Loyd where all blocks are single squares and the initial position is randomly chosen (the blocks are shuffled prior the ordering attempt) <sup>4</sup>.

In order to reuse the same script for different initial and final positions, I make the script load the puzzle description from a txt file like this:

---

<sup>4</sup>Nomignolov already comes with the script to play this puzzle. See Section 3.3

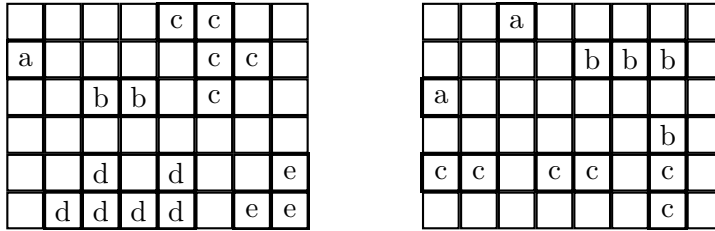


Figure 2: Examples of polyminoes and pseudo-polyminoes

```

aabbcc
dde.f
dde.g
hhij
.....
...dd
...dd
.....

```

Figure 3: Txt file with puzzle's description

I didn't want to make the script heavy by adding checks to this files. Nomignolov interprets each dot `.` as an empty cell, each `#` as an inaccessible wall, and each letter `[a-z]` as a piece of one of 26 different pieces. The text before the empty line is the initial position, the text after it is the final position.

The file from Figure 3 describes the puzzle of Figure 4.

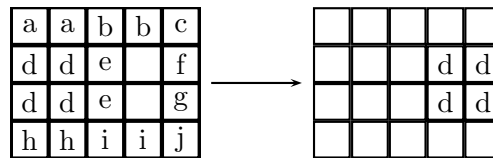


Figure 4: Level SBP\_001.txt

**Variants.** The script handles puzzles loading a txt. But which file should be loaded? The current version 1.2 “Benbow” of Nomignolov does not handle variants, so I added this feature and it will be present in version 1.3.

From the script's point of view, a variant is just a string, and at the

beginning of the file all possible variants are listed in a vector:

```
variants = {"SBP_001.txt",  
           "SBP_002.txt"}
```

Nomignolov reads this vector and, if present, shows the list of variants to the player. The game has an identification number for the selected variant. This number can be accessed within the script, just as happens for the game's flag, by mean of a couple of functions:

```
get_variant()  
set_variant()
```

The variants vector is the list of variants' names, but for laziness I used a list of filenames. The variant is like a parameter given to the script, and could be used to games as well. As a simple example, the `victory` function could read the variant and manage the "misère" version of a game, where a player wins by loosing with standard rules. There are countless many use for this flag: managing board dimensions, number of pieces, handicaps, progressions, etc.

### 3.2 Editor script

The real objective of writing this script is to show Nomignolov capabilities. Indeed, the editor is usable to write txt files interpreted by the Sliding block puzzle script.

The script shows a  $14 \times 14$  board; the first column is the menu of the editor.

The editing phase is splitted in drawing the initial position and drawing the final position. Each phase is a set of actions (turns) that consists on selecting a type of piece or place a piece of that type on the board.

Pieces are selectable by clicking on them from the menu. Among the pieces there is one (with a circle on it) representing the wall and one (with a cross on it) used to remove previously placed pieces. Two additional pieces (with arrows on them) can be clicked to change the set of 8 pieces currently displayed <sup>5</sup>.

The `PASS` button must be pressed when the initial phase is finished, and must be also pressed when the final position is completed.

In the second phase, a red mark is positioned over each piece from the initial position. In the final position, pieces can be placed only over marks, or over empty cells, and obviously just of the types already present in the initial position!

The editor may be in a number of *states*, according to which is the current phase, and which type of piece is selected. States has a unique identification

---

<sup>5</sup>The arrow pieces are no moveable pieces. They are just like buttons!

number, which is written and read in the *flag* of the game, accessible via the `set_flag` and `get_flag` functions.

### 3.3 Comments

**Solving puzzles.** Nomignolov is not yet equipped with a dedicated puzzles' solver. The program allows for any number of players, and the only thing it can do right now is to apply a kind of minimax algorithm where the current player can change at any depth of the game's tree (and remains the same for puzzles).

For the 14-15 puzzle, I wrote a simple evaluation function that computes the king distance (aka Manhattan or pedestrian distance) of each tile from the current position to the final one: the less the sum of the distances of all the pieces the best.

In this puzzle, there are 4 possible moves per turn (3 in the sides and 2 in the corners). The branching factor of the game's tree is thus extremely small. Unluckily, to solve the puzzle, the analysis should be very depth: there are a lot of local minimums for the sum of distances, and a lot of bad moves should be made as to reach the global one (the winning position).

I solved this problem by adding the constraint that a move could not be undone. This reduces even more the branching factor, allowing for greater depth. And, a lot more important, does not allow to be captured in a local minimum. With this heuristic, Nomignolov solves the puzzle.

**Sliding block puzzles.** For the general version of the puzzle I forbid the direct undo of a move if there are other moves possibilities. In fact, it could happen that a move doesn't leave other moves and must be undone.

This constraint is not enough, because of the general less mobility of pieces, and because generally only some pieces have a destination.

I don't think that there exists an evaluation function of the game situation able to solve a general puzzle with the minimax algorithm. You are wellcome to disprove me by mailing me your algorithm! I should write a dedicated algorithm for scripts with just one player. Not that my minimax is the state of the art!

## 4 Conclusions

Playing with Nomignolov allows me to find and add features that are usefull. Anyway, I don't want to add unnecessary things to the program, but rather show how the existing features could be used in a lot of ways.

Nomignolov hankers for scripts and, in the true spirit of the internet, it aims to be a collaborative effort. If you are interested in writing scripts, you can read the documentation which comes in the zip file. And if you have any doubt, fell free to drop me an e-mail, and I'll reply gladly.

Bye!