## Intruduction

The game *HuaRongDao* belongs to a family of puzzles known as brain teasers. [1] This name comes from the well known story that General Kwan Kung releasing his enemy, King Cao Cao, at Huarongdao in a critical battle, which happens during the "Three-Kingdom Period" in China (184 - 286 AD).

In this block-rearranging puzzle, two distinguishing blocks, the $2 \times 2$ block and the $2 \times 1$ one denote Cao Cao and Kwan Kung respectively; whereas other blocks denote other generals ($1 \times 2$ blocks) or soldiers ($1 \times 1$ blocks). The goal of the puzzle is to move the biggest block to the exit at the bottom of board, and this represents that Cao Cao flees from the surrounding of the enemies successfully.

## Implemetation

The program includes two modules: the puzzle solver (*solver.hs*, 700 lines Haskell code) and the graphics interface of the game (*game.hs*, 300 lines Haskell code).

- The solver uses the *Breadth-First Search* to get the optimal solution of the puzzle. Since the solution needs 50-100 steps normarlly, the algorithm needs some efficient data structures to store the search space. The current implementation adopts a kind of *Binary Search Tree*, the AVL tree, as the data structure to store the visited nodes in the search space. This representation can solve the puzzles with an optimal solution around 50 steps. However, the memory provided by Haskell can hardly afford the puzzles which need over 100 steps search for the solution. Hence the implementation provides a *compact* reprenstation of the configurations, which can solve the hard instances (for example, the $startSt1$ puzzle which needs 111 steps to get to the goal configuration). Note that this representation introduces some overhead for the translation as a trade-off. [2]
- The GUI is trivial for this problem. The only thing need to do is to get the solution from the *solver* module, and to show the moves step by step on the display. One possible improvement is to extend the GUI to provide an interactive game, which enables the users to play the game by themselves. This is not very hard to implement, based on the current implementation.

## Usage

The text-only puzzle solver can be used in hugs:

```
Hugs> :l solver
HRDsolver> solveHRD start okStatus
```

The graphics solver needs to use rugs:

```
Rugs> :l game
HRDgraphics> main
```

To select different initial configurations, the user can set the variable *start* to different values (the preset values are normally named $startXXX$), in *solver.hs*.

[1] The puzzle is also called *Red Donkey* in some other countries.

[2] This trade-off seems reasonable since the maximum possible search depth is 132, and almost all of the solvable configurations can be solved within several minutes, including the time for encoding and decoding.

**References**

[1] G. Cousineau and M. Mauny, *The functional approach to programming*, Cambridge University Press, 1998.
[2] P. Hudak, *The Haskell School of Expression: learning functional programming through mutltimedia*, Cambridge University Press, 2000.
[3] P. Hudak, J. Peterson, and J. H. Fasel, A gentle introduction to Haskell 98, 1999.
[4] A. Reid, The hugs graphics library, 2000.
[5] C. Shurmann, Lecture notes for CPSC529, Department of Computer Science, Yale University, 2001.
[6] The Haskell project homepage, *http://www.haskell.org*.