SOEN6441
KenKen Puzzle (Common Lisp)

Current Version: 0.1
Date: June 13, 2012
Author: Yuan Tao
Instructor: Dr. C. Constantinides

Concordia University

Table of Contents

Table of Figures

1. About the KenKen Puzzle

You can play and find much useful information about KenKen puzzle from the following website: http://www.kenken.com/.

Here is the layout of a 3*3 game which is developed with Common Lisp:

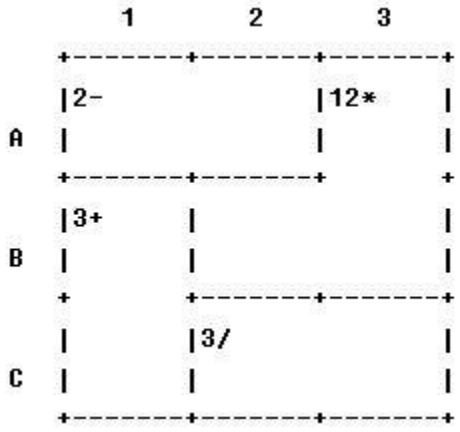

Figure 1-1 Sample layout of the game

From the figure, the game consists of four cages. Each of the cages is made up of several cells. Within the cage, there are a number and an operator. You might have guessed that the number of the cage should be a combination of the operator and the values of the cells within the cage. Another rule of the game is that the value of the cell should not greater than the size of the game (for above 3*3 game, the size is 3), and each line or column should not have duplicated values.

2. System Design

2.1. Data structure

2.1.1. Layout of the game

To make the development of the game more extendable, we need to define a format of how to describe the game with the syntax of Common Lisp. Here, the List is used. The following example is the list format of the above figure:

'((GAME3 3 ((2 - A1 A2) (12 * A3 B2 B3) (3 + B1 C1) (3 / C2 C3))))

Generally, the meaning of the list structure is like the following figure:

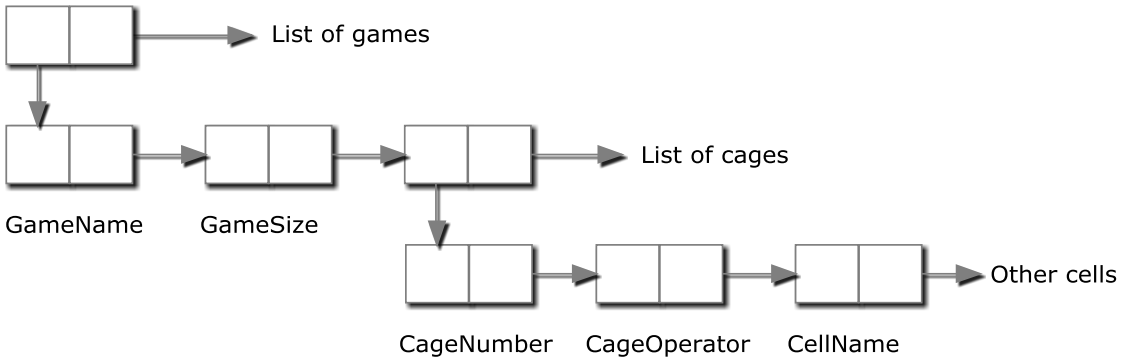

Figure 2-1 List structure of the game

For the first version, all the games that the user can select are hard coded as the structure described above. For later versions, if possible, we can store the game in files or let the program generate different game layouts automatically. These future features related to the games generation will not affect the current architecture of the game.

2.1.2. Values of cells

Besides the layout of the game, there is also another data structure to store the values of all cells for the current game which the user is playing:
'((a1 1) (a2 2))

It is a list. Its element is a list, which is made up of the name of the cell and the value of the cell.

2.2. User interface

To make the development of the game relatively simple as well as not affect the friendliness of the user interaction, we do not use GUI. Instead, we decide to use Console UI.
As a result, the game should be drawn line by line and character by character, just like Figure 1-1.

Another key interaction between the user and game is that the user needs to tell the program the value of the each cell. The format of the user input is like this:
CellName=Value CellName=Value …

The program continues asking user input the cell values until all cells get a value.
Here is an example about how to get user input:

```
Please set values to cells (e.g. a1=1 b2=2 a1=2) (type q to exit game):
a1=2 a2=1 a3=3 b2=2
Game Name: GAME3
       1       2       3
   +-------+-------+-------+
   |2-             |12*    |
 A |   2       1   |   3   |
   +-------+-------+       +
   |3+     |               |
 B |       |   2           |
   +       +-------+-------+
   |       |3/             |
 C |       |               |
   +-------+-------+-------+
Please set values to cells (e.g. a1=1 b2=2 a1=2) (type q to exit game):
```

Figure 2-2 Example of setting cell values

3. Unit Test

3.1. Test Framework

Here we use the test framework provided by chapter 9 of the book 'Practical Common Lisp' written by Peter Seibel, 2005.

```
(defvar *t-name* nil)
(defun report-result (result form)
  (format t "~:[FAIL~;pass~] ... ~a: ~a~%" result *t-name* form)
  result)
(defmacro with-gensyms ((&rest names) &body body)
  `(let ,(loop for n in names collect `(,n (gensym)))
     ,@body))
(defmacro check (&body forms)
  `(combine-results
     ,@(loop for f in forms collect `(report-result ,f ',f))))
(defmacro combine-results (&body forms)
  (with-gensyms (result)
    `(let ((,result t))
       ,@(loop for f in forms collect `(unless ,f (setf ,result nil)))
       ,result)))
(defmacro deftest (name parameters &body body)
  `(defun ,name ,parameters
     (let ((*t-name* ',name))
       ,@body)))
```

4. How to run

4.1. File list

- main.cl
  Source code
- reame.pdf
  This file.

All the files you can get from:
https://all-projects-concordia.googlecode.com/svn/soen6441

4.2. Compile error with Allegro CL Free Express Edition 8.2

If you compile the source code with Allegro CL Free Express Edition 8.2, you might get an error:
Error: attempt to call `WITH-GENSYMS' which is an undefined function.

(**Note that** this error will NOT happen if you compile with LispWorks 6.0 Personal.)

In this case, please do as the following steps:
1. Backup the definition of combine-results macro.
2. Delete the body of combine-results macro, like this:
   (defmacro combine-results (&body forms))
3. Compile (Ignore the warnings).
4. Restore the macro definition.
5. Compile again.

4.3. Run the program

Call the following function on the listener:
(main)

4.4. Run the test cases

(test-all)
If any case fails, the return of the function is NIL; otherwise, you will get T.

Appendix A: Revision History

Version	Date	Author	Remark
V0.1	June 13, 2012	Yuan Tao	Draft