Sudoku : Sudoku Domain Objects

This page last changed on Jun 15, 2009 by nbasham.

⚠ The following is a work in progress.

Overview

A Sudoku board consists of 81 cells organized in 9 rows by 9 columns. Each cell has a unique index from 0 to 80 start from left to right and top to bottom. The board can also be described in terms of 9 grids each 3 rows by 3 columns starting from left to right, top to bottom.

The goal of Sudoku is for the user to fill in each cell with the correct value. A value is a number between 1 and 9 defined by the problem. The Sudoku game starts off with a number of cells revealed to the user, the harder the level the fewer cells that are initially revealed. The user can then enter a guess or hints in each cell until all cells have values. The game is won when each cell has a value and each of those values is unique to it's row, column and grid. The user's guess, hints and initially revealed cells make up the user's solution, note the solution may or may not be correct.

The Board

81 cells, made from 3x3 collection of grids, consisting of 9 rows and 9 columns.

	1		6		7			4
	4	2						
8	7		3			6		
	8			7			2	
			8	9	3			
	3			6			1	
		8			6		4	5
						1	7	
4			9		8		6	

Cells

81 cells make up the board.

Rows And Columns

A board consists of rows, each made up of 9 horizontal cells that span 3 grids and columns, each made up of 9 vertical cells that span 3 grids.

Grids

A 3x3 collection of cells starting from left to right, top to bottom.

0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8

Cell Index

Each cell is uniquely identified by a index from 0 to 80, starting at the top left of the board and proceeding from left to right and top to bottom.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	1	2	3	4	5	6	7	8
1	9	10	11	12	13	14	15	16	17
2	18	19	20	21	22	23	24	25	26
3	27	28	29	30	31	32	33	34	35
4	36	37	38	39	40	41	42	43	44
5	45	46	47	48	49	50	51	52	53
6	54	55	56	57	58	59	60	61	62
7	63	64	65	66	67	68	69	70	71
8	72	73	74	75	75	77	78	79	80

The Problem

81 item array with the correct values for each cell. Each item can have a value between 1 and 9.

An example problem:

```
864531297951627843273489615147956382396248571582173964625814739719365428438792156
```

Revealed Indexes

A list of indexes that determine which cells will have their values revealed at the start of the puzzle. These values are not editable. Each item represents the starting visibility for the corresponding problem item. More difficult puzzle levels initially reveal fewer problem items.

Level	**Easy**	**Medium**	**Hard**	**Chanllenger**
Problems revealed	35	30	27	24

The Solution

81 item array that contains the current state of the Sudoku board. This is made up of the revealed problem cells, user guesses, user hints, and zeros for all unanswered values.

Guesses

User guess for a cell, valid values are integers between 1 and 9.

value	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
guess	1	2	3	4	5	6	7	8	9

Hints

User marker for a cell, the value is between 1 and 9 and there can be 0 to 9 hints per cell. Hints are used as clues by the user for possible guess values.

value	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
hint	-1	-2	-4	-8	-16	-32	-64	-128	-256

Values

The numeric representation of a cell's value.

Empty cells (those with no guess or hints) have a value of 0. A cell can contain 0 to 1 guesses and 0 to 9 hints. If a guess is entered on a cell containing hints, the hints are removed. If a hint is entered on a cell containing a guess, the guess is removed.

value	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
guess	1	2	3	4	5	6	7	8	9
hint	-1	-2	-4	-8	-16	-32	-64	-128	-256

User enters a guess of 4, the cell value is 4.
User enters a hint of 4 into a cell, the cell value is -8.
User enters hints of 4 and 6 into a cell, the cell value is -8 + -32 = -40.

Interpreting cell values goes as follows:

```
if a cell has value of 0 it is empty
else if a cell has a positive value it contains a [#guess] of that value
else the cell is negative
    for n = 1 to 9
        hasHint = -n & (1 >> n)
```

Putting It All Together

Ind	0	1	2	3	4	5	6	7	8	....	72	73	74	75	75	77	78	79	80
Board	8	6		45	3	1		9	137	....	4	3	8	7	9		1		26
Problem	8	6	4	5	3	1	2	9	7	....	4	3	8	7	9	2	1	5	6
Value	8	6	0	-24	3	1	0	9	-69	....	4	3	8	7	9	0	1	0	-34

Sudoku Model

Local Model

User State

Type	Variable	Description
integer	level	The current default level
long	gameID	The game id of the last game or the game in progress
integer array	solution	The board state
boolean	showConflicts	Preference: Alert user when a guess conflicts with the board state. Note a correct answer could conflict
boolean	showTimer	Preference: Display elapsed time
GameRecord array	history	The users history of game results.
integer	easyIndex	The last easy level puzzle id started.
integer	mediumIndex	The last medium level puzzle id started.
integer	hardIndex	The last hard level puzzle id started.
integer	challengerIndex	The last challenger level puzzle id started.

GameRecord

Type	Variable	Description
long	puzzleId	A unique id the identifies a puzzle

long	date	The date the puzzle was started
long	elapsedTime	The amount of time the user has spent on a puzzle.
boolean	paused	Did the user pause or leave the game.
integer	leve	Game level i.e. Easy, Medium, Hard or Challenger. This information may be part of a PuzzleRecord, but that may or may not be remotely stored.

Remote Model

Device

Type	Variable	Description
string	deviceId	Universally unique device id

PuzzleRecord

Type	Variable	Description
long	puzzleId	A unique id the identifies a puzzle
integer array	problem	Problem.
integer	level	Easy, medium, hard or challenger.

Sudoku API

```
public class Puzzle {
    long id;
    int index;
    int level;
    int[81] problem;
    int[] mask;
}

public class Board {
    void initialize(Puzzle puzzle);
    boolean solved();
    int[81] getSolution();
    void setSolution(int[81] solution);

    int getValue(int index);
    void setValue(int value, int index);
    boolean conflict(int value, int index);
    boolean hasHint(int index);
    void clearHint(int value, int index);
    void clearHints(int index]);
  }
```