Projects 5 and 6: Sudoku

Part 1: Constraint propagation

Part 2: Back-track search (recursive guess-and-check)

Sudoku

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

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

Each row, column, and 3x3 sub-square must contain each of the digits 1..9

Two part project:

Week 5:

Constraint propagation to solve simple puzzles and recognize correct solutions

"Naked single" and "hidden single" tactics

Week 6:

Search for a solution to harder puzzles

Depth-first recursive search with back-tracking

Check & Help

Is board ok so far?

Check uniqueness of symbols in each row, column, and block (ignoring unfilled cells)

Help with unfilled (“.”) cells:

Allowed symbols in a tile is intersection of

symbols not yet used in row

symbols not yet used in column

symbols not yet used in square

Two tactics: Naked single, hidden single

Sudoku help: Partial boards

We will use the "sadman sudoku" format:

"." marks a position that has not been chosen

See

http://www.sadmansoftware.com/sudoku/faq19.htm

(We will use the minimal puzzle format.

Displaying or printing "pencil marks" would be a nice extension.)

We can find standard solution tactics on the web ...

SadMan Software

Sudoku Technique - Naked Single

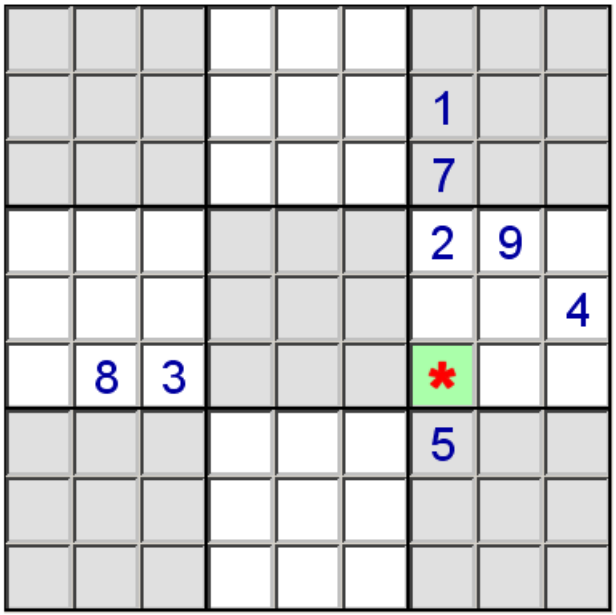
Home | Applications | Delphi | Free Stuff | Support Forums | Everything Else

Home > Sudoku > Solving Techniques > Naked Single / Singleton / Sole Candidate

Naked Single (Singleton, Sole Candidate)

Whilst solving Sudoku puzzles, it is often the case that a cell can only possibly take a single value, when the contents of the other cells in the same row, column and block are considered. This is when, between them, the row, column and block use eight different digits, leaving only a single digit available for the cell.

For example, in the partial Sudoku puzzle below, the marked cell can only be a 6. All other digits are excluded by the contents of the other cells in the row, column and block.



... and translate to design, then pseudocode, then code.

"Naked single" (aka "sole candidate")

Human tactic:

Start with 9 candidates in each cell

Cross off candidates used in the row, column, and block

If there is only one candidate left in a cell, choose it

Program tactic:

Each cell has a "candidates" attribute, a Python set

For each row, column, and block

- Loop through the group to find used symbols
- Loop again to remove used symbols from "possible"
- If len(possible) is 1, choose that symbol

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

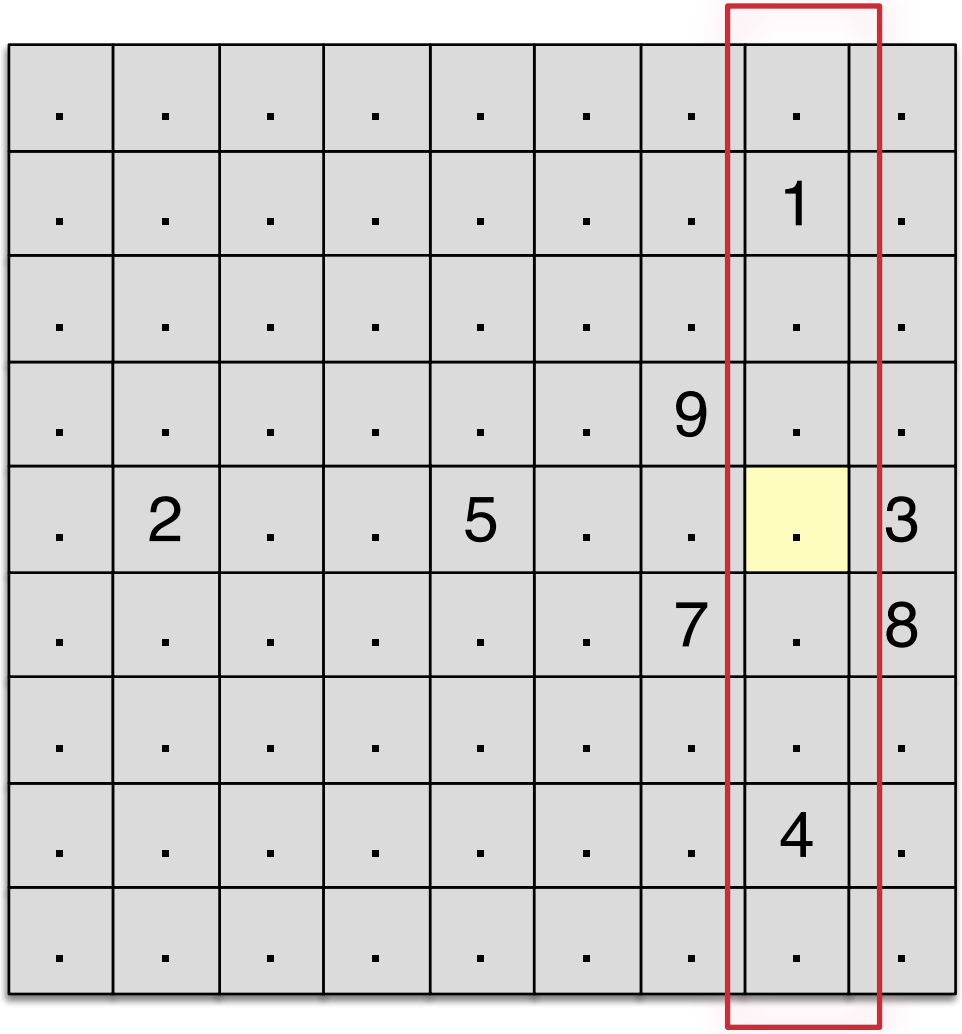
Candidates

1	2	3	4	5	6	7	8	9

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

Candidates

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



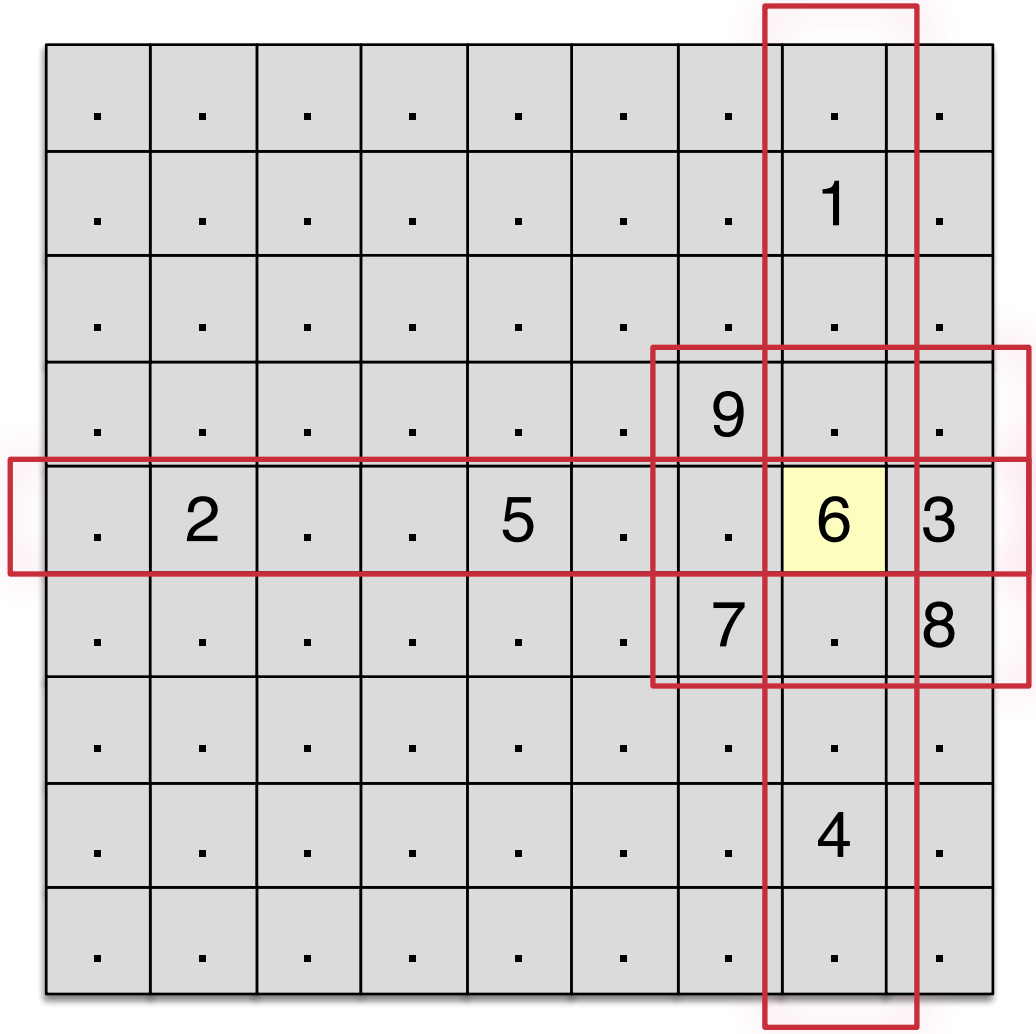
Candidates

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

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

Candidates

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



Candidates

Keeping an invariant

It helps to establish an object invariant:

At all times, the Tile.candidates includes all possible choices for the Tile object

Tile.candidates may become smaller (lose elements) when we discover constraints, but it never becomes larger during constraint propagation

If tile.candidates has only one element, then tile.symbol is that element

Every method and function maintains the invariant*

(*for now … until we do back-track search)

Pseudocode: Maintain invariant while reducing the "candidates" sets

Program tactic: "naked single" or "sole candidate"

- Each Tile has a "candidates" attribute, a Python set
- For each row, column, and block
  - Loop through the group to find used symbols
  - Loop again to remove used symbols from "possible"
  - If len(possible) is 1, choose that symbol

I want to avoid writing the “check unique entries” code three times

- Once looping through elements in a row
- Once looping through elements in a column
- Once looping through elements in a square

DRY: “Don’t Repeat Yourself”

How can I factor that code out?

I want to avoid writing the “check unique entries” code three times

- Once looping through elements in a row
- Once looping through elements in a column
- Once looping through elements in a square

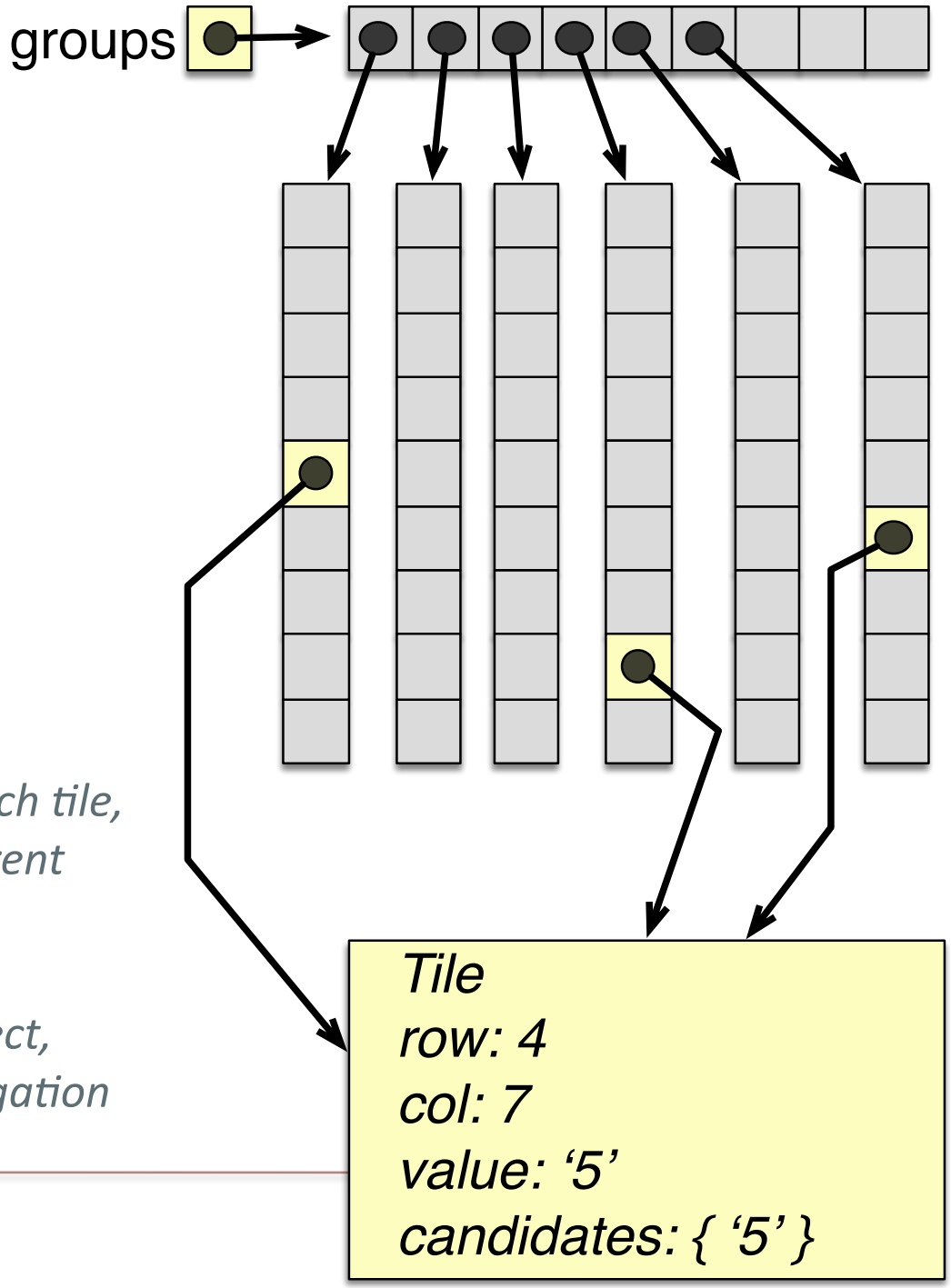
DRY: “Don’t Repeat Yourself”

How can I factor that code out?

Write it just once, for a list of 9 elements

Board provides the ***same*** **tiles** in **different lists**, by row, column, and square

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

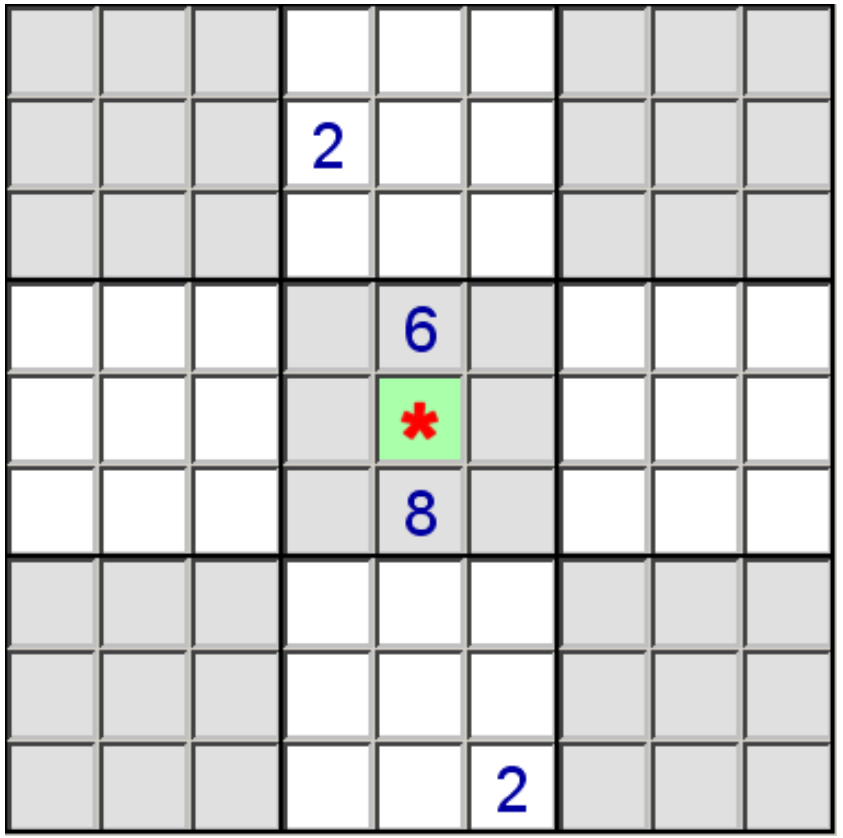


Board contains only one copy of each tile, but each tile appears in three different groups.

Each group is actually a Group object, with methods for constraint propagation

Hidden single tactic

A 2 has got to go somewhere in that middle block.
Can't go in first column.
Can't go in third column.
Must go in the only place it can.



http://www.sadmansoftware.com/sudoku/hiddensingle.htm

Let's turn this into design
and pseudocode

(Trickier than naked single,
and works only in combination with
reduction of candidates sets in naked single.)

Hidden single (pseudocode)

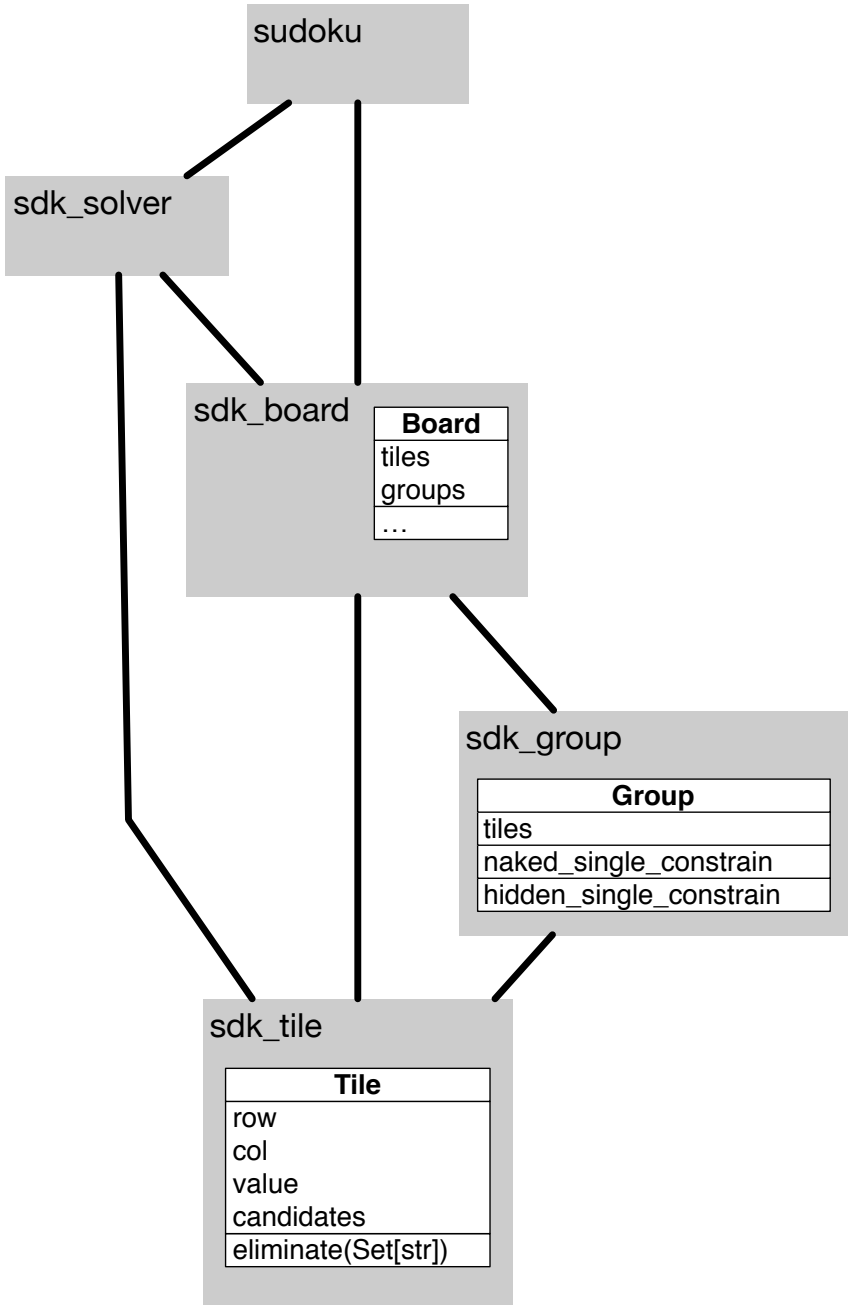
```
LEFTOVERS =  { CHOICES }
for tile in group:
      LEFTOVERS -= { tile.value}

for value in LEFTOVERS:
     places to put value = [  ]

     for tile in group:
           can value go in tile?
           if it can go there,
               add to places to put value

    if len(places to put value) == 1:
        put the value there!
```

Notes:
In Python, an empty set is created by set(), and a set containing items in a list li is created as set(li). Curly braces are not used because curly braces indicate a dict (and dicts came first; sets are a newer feature in Python). A set containing a single element x would be set([x]), that is, the set containing the elements in a list that just contains x.

"can value go in tile" is actually a method in the Tile object, although it has a shorter name.

"put the value there" is also a method of Tile, also with a shorter name.

Code organization (model part)



not shown: main graphical view in sdk_display.py, additional text view in sdk_debugview.py, MVC plumbing in events.py

And of course a test suite …