

# Package ‘RSDK’

March 12, 2022

**Type** Package

**Title** Sudoku with R

**Version** 1.0.1

**Author** EL KHMISSI Mohamed

**Maintainer** EL KHMISSI Mohamed <mohamed.el-khmissi01@etu.umontpellier.fr>

**Description** This is a sudoku game package with a shiny application for playing .

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**Imports** testthat (>= 3.0.0), graphics, grDevices, shiny, shinyWidgets,  
keys

**Config/testthat/edition** 3

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-03-11 23:20:05 UTC

## R topics documented:

atbox	2
atcol	2
atrow	3
bt_solver	4
check_grid	4
grid_gen	5
grid_gen_cplt	5
grid_gen_lv	6
ispossible	6
nbrposs	7
order_wposs	8
perm_mat	8
perm_vec	9

plt_grid . . . . .	9
plt_grid_play . . . . .	10
poss . . . . .	10
runSudoku . . . . .	11
solver . . . . .	11

<b>Index</b>	<b>12</b>
--------------	-----------

---

atbox	<i>atbox()</i>
-------	----------------

---

### Description

This function checks if a value already exists in a 3 by 3 box from a sudoku grid

### Usage

atbox(x, i, j, n)

### Arguments

x	A sudoku grid
i	An index of a line from the box
j	An index of a column from the box
n	a value to check its existance in the box that contains the cell of the index (i,j)

### Value

TRUE if the checked value is on the box or FALSE if the checked value is not on the box

### Examples

```
atbox(x=grid_gen(49),1,4,8)
```

---

atcol	<i>atcol()</i>
-------	----------------

---

### Description

This function checks if a value already exists in a column from a sudoku grid

### Usage

atcol(x, j, n)

**Arguments**

x	A sudoku grid
j	An index of a column from the grid
n	a value to check its existence in the column j

**Value**

TRUE if the checked value is on the column or FALSE if the checked value is not on the column

**Examples**

```
atcol(x=grid_gen(63),1,8)
atcol(x=grid_gen(49),7,6)
```

---

atrow	<i>atrow()</i>
-------	----------------

---

**Description**

This function checks if a value already exists in a row from a sudoku grid

**Usage**

```
atrow(x, i, n)
```

**Arguments**

x	A sudoku grid
i	An index of a row from the grid
n	a value to check its existence in the row i

**Value**

TRUE if the checked value is on the row or FALSE if the checked value is not on the row

**Examples**

```
atrow(x=grid_gen(63),1,8)
atrow(x=grid_gen(49),7,6)
```

bt\_solver                    *bt\_solver()*

---

**Description**

This function is a recursive function that solves a sudoku grid using the backtracking algorithm

**Usage**

```
bt_solver(x)
```

**Arguments**

x                    A sudoku grid

**Value**

A list of two elements in the first one there is the grid x solved as a matrix of 9 by 9, and the second one contains the number of backtracking does R do to solving it.

**Examples**

```
bt_solver(x=grid_gen(49))
```

---

check\_grid                    *Check\_grid()*

---

**Description**

This function checks if a 9 by 9 grid is a complete sudoku grid (each number appear only once in its row, column and box )

**Usage**

```
check_grid(x)
```

**Arguments**

x                    A sudoku grid

**Value**

True if x is a complete sudoku grid False if x is not

**Examples**

```
check_grid(x=grid_gen_cplt())  
check_grid(x=grid_gen(54))
```

---

grid_gen	<i>grid_gen()</i>
----------	-------------------

---

**Description**

This function generates a sudoku grid with a given number for the empty cells

**Usage**

```
grid_gen(t)
```

**Arguments**

t	The number of the empty cells
---	-------------------------------

**Value**

A sudoku grid with t empty cells

**Examples**

```
Grid_45 = grid_gen(45)
```

---

grid_gen_cplt	<i>grid_gen_cplt()</i>
---------------	------------------------

---

**Description**

This function generates a complete sudoku grid randomly

**Usage**

```
grid_gen_cplt()
```

**Value**

A complete sudoku grid

**Examples**

```
Grid_complete = grid_gen_cplt()
```

---

grid_gen_lv	<i>grid_gen_lv()</i>
-------------	----------------------

---

### Description

This function generates a sudoku grid for four levels of playing "Easy", "Difficult", "Hard" and "Legend" based on the number of backtraking does the finction [bt\\_solver](#) did to solve the grid.

### Usage

```
grid_gen_lv(lv)
```

### Arguments

lv	A string argument level for the grid and must be "Easy", "Difficult", "Hard" or "Legend"
----	--

### Value

A sudoku grid associate to the level in lv

### Examples

```
grid_gen_lv("Easy")
grid_gen_lv("Legend")
```

---

ispossible	<i>ispossible()</i>
------------	---------------------

---

### Description

This function checks if it is possible to put a given number in a given empty cell

### Usage

```
ispossible(x, i, j, n)
```

### Arguments

x	A sudoku grid
i	The index of the row of the given cell
j	The index of the column of the given cell
n	The number that we want to check if is possible to put it in the cell of the index (i,j)

**Value**

True if it is possible to put n in the cell (i,j)

**Examples**

```
ispossible(x=grid_gen_cplt(),4,5,6)
ispossible(x=grid_gen_cplt(),4,5,6)
```

---

nbrposs	<i>nbrposs()</i>
---------	------------------

---

**Description**

This function returns the number of possibilities for a given empty cell

**Usage**

```
nbrposs(x, i, j)
```

**Arguments**

x	A sudoku grid
i	The index of the row of the given cell
j	The index of the column of the given cell

**Value**

Number of possibilities for the cell (i,j)

**Examples**

```
nbrposs(x=grid_gen_cplt(),5,7)
nbrposs(x=grid_gen_cplt(),6,9)
```

---

order_wposs	<i>order_wposs()</i>
-------------	----------------------

---

**Description**

This function returns an ordred data frame by number of the possibilities for all the empty cells in the grid with index of row for the first column and index of column for the second column and the number of possibilities in third column

**Usage**

```
order_wposs(x)
```

**Arguments**

x	A sudoku grid
---	---------------

**Value**

data frame

**Examples**

```
order_wposs(x=grid_gen_cplt())
```

---

perm_mat	<i>perm_mat()</i>
----------	-------------------

---

**Description**

This function permutes the columns of a given matrix with a cyclic permutaion

**Usage**

```
perm_mat(a, v)
```

**Arguments**

a	A matrix
v	The length of the cyclic permutation

**Value**

A matrix permuted cyclically by v columns

**Examples**

```
perm_mat(a=diag(1,5),4)
```

---

perm_vec	<i>perm_vec()</i>
----------	-------------------

---

**Description**

This function permutes a given vector with a cyclic permutation

**Usage**

```
perm_vec(x, i)
```

**Arguments**

x	A vector
i	The length of the cyclic permutation

**Value**

A vector permuted cyclically by x values

**Examples**

```
perm_vec(1:6,4)  
perm_vec(27:50,15)
```

---

plt_grid	<i>plt_grid()</i>
----------	-------------------

---

**Description**

This function plots a given sudoku grid

**Usage**

```
plt_grid(X)
```

**Arguments**

X	A sudoku grid
---	---------------

**Value**

a plot of the grid

**Examples**

```
plt_grid(X=grid_gen_cplt())
```

plt\_grid\_play            *plt\_grid\_play()*

---

**Description**

This function gives a reactive plot of the grid for the shiny application

**Usage**

```
plt_grid_play(B, x)
```

**Arguments**

B	Initial grid
x	The grid that the user put the numbers on it

**Value**

a plot of the grid with the user input with a different color red if the input is on the wrong cell and green if the input is on the right cell

---

poss                    *poss()*

---

**Description**

This function returns a vector of possibilities for a given empty cell

**Usage**

```
poss(x, i, j)
```

**Arguments**

x	A sudoku grid
i	The index of the row of the given cell
j	The index of the column of the given cell

**Value**

Vector of possibilities for the cell (i,j)

**Examples**

```
poss(x=grid_gen(46),4,7)
poss(x=grid_gen(49),3,9)
```

---

runSudoku	<i>runSudoku()</i>
-----------	--------------------

---

**Description**

runSudoku()

**Usage**

runSudoku()

**Value**

Opens the sudoku shiny application

---

solver	<i>solver()</i>
--------	-----------------

---

**Description**

This function is a recursive function that solves a given sudoku grid for shiny application and it is more optimized than the backtracking solver on the function [bt\\_solver](#)

**Usage**

solver(x)

**Arguments**

x                    A sudoku grid

**Value**

The grid x solved

**Examples**

```
solver(x=grid_gen(46))
```

# Index

atbox, [2](#)  
atcol, [2](#)  
atrow, [3](#)  
  
bt\_solver, [4](#), [6](#), [11](#)  
  
check\_grid, [4](#)  
  
grid\_gen, [5](#)  
grid\_gen\_cplt, [5](#)  
grid\_gen\_lv, [6](#)  
  
ispossible, [6](#)  
  
nbrposs, [7](#)  
  
order\_wposs, [8](#)  
  
perm\_mat, [8](#)  
perm\_vec, [9](#)  
plt\_grid, [9](#)  
plt\_grid\_play, [10](#)  
poss, [10](#)  
  
runSudoku, [11](#)  
  
solver, [11](#)