Interactive Sudoku

Curtis Bright, Maplesoft

Restart Check Solve

New Game:

Random From File From Web

Instructions

- Select "Yes" after opening the worksheet, or press the !!! button, to run the code to start the game (or see the "implementation details" to study and manually run the code).
- The **Restart** button clears the entries that were entered by the player.
- The **Check** button checks if it is possible to fill the remaining entries in a consistent way (every row, column, and block contains distinct digits).
- The **Solve** button fills the remaining entries in a consistent way if it is possible to do so. The check and solve functions are done by reducing the problem to SAT and calling Maple's built-in SAT solver as described in the "implementation details".
- The **Random** button randomly generates a new Sudoku puzzle with a unique solution. The generation is done using a SAT solver as described in the "implementation details".
- The **From File** button loads a new Sudoku puzzle from a Sudoku sdk file selected by the player.
- The **From Web** button loads a new Sudoku puzzle of easy difficulty using an online source.

Implementation details

- The game uses Maple's plotting commands to draw the board, Maplets for the dialog boxes, and embedded components for the buttons.
- No knowledge of Sudoku solving or generation algorithms were used in the implementation. Instead, the rules of Sudoku were encoded into Boolean logic and Maple's built-in SAT solver was used to automatically generate and solve Sudoku puzzles. See the Satisfy command for more information about SAT solvers.
- The source code of the game may be viewed by executing the following line of code:

```
1  DocumentTools:-SetProperty(InteractiveSudokuCode,
   visible, true):
```

Encoding the rules of Sudoku

- To encode the rules of Sudoku in Boolean logic we use the Boolean variables $S_{i,j,k}$ with $1 \leq i, j, k \leq 9$ to denote that the square at (i, j) contains the digit k.
- The rules of Sudoku state that each square must be filled with a digit between 1 and 9 and that the same digit cannot appear twice in the same row, column, or block. The first constraint has the form $S_{i,j,1} \vee S_{i,j,2} \vee \cdots \vee S_{i,j,9}$ for each $1 \leq i, j \leq 9$ and the second constraint has the form $S_{i,j,k} \Rightarrow \neg S_{i',j',k}$ where $1 \leq i, j, i', j', k \leq 9$ and (i', j') does not equal (i, j) but is in the same row, column, or block as (i, j).

The Check and Solve implementations

- In addition to the constraints encoding the rules of Sudoku, the **Check** and **Solve** implementations call the SAT solver with the unit clauses $S_{i,j,k}$ where the square at (i, j) contains the digit k in the current board's state.
- The **Check** implementation just calls Satisfiable to determine if some solution exists. The **Solve** implementation calls Satisfy and if a satisfying assignment is found then for each $S_{i,j,k}$ that has been assigned to true the square at (i, j) is filled with the digit k.

The Random implementation

- The **Random** implementation begins by calling the SAT solver with the clauses encoding the rules of Sudoku but no clauses encoding the starting configuration. The result produced by the SAT solver gives a completed Sudoku grid R (a random seed is passed to the SAT solver so that a different R is produced each time). A random ordering of the entries is generated and the first 50 entries are selected as the potential starting configuration of a Sudoku puzzle. This puzzle has the solution R by construction, though other solutions may also exist.
- To verify that the generated solution is unique, we re-run the SAT solver with the additional 50 unit clauses corresponding to the starting configuration along with the constraint $\bigvee_{R[i,j]=k} \neg S_{i,j,k}$ which blocks the solution R. If the SAT solver returns another solution then

we start over and find a new R to try. Otherwise the first 50 entries of R form a legal Sudoku puzzle.

- Additionally, it may be the case that we could use fewer than 50 entries and still obtain a Sudoku puzzle with a unique solution. To estimate how many entries we need to assign using only a few extra calls to the SAT solver we define $lo := 20$, $hi := 50$, $mid := \text{round}((lo + hi)/2)$ and repeat the last step except using only the first mid entries of R. If the resulting SAT instance is satisfiable then we need to use more than mid entries to ensure a unique solution and if the resulting SAT instance is unsatisfiable then we can perhaps use fewer than mid entries. Either way we improve the bounds on how many entries to assign (in the former case we can update lo to mid and in the latter case we can update hi to mid) and this step can be repeated a few times to find more precise bounds on how many entries need to be assigned to ensure a unique solution exists.