Générateur de Sudoku

Ce notebook wxMaxima permet de générer une grille de Sudoku ainsi que sa solution. Il suffit d'exécuter les cellules ci-dessous pour obtenir une grille de Sudoku à compléter.
Pour obtenir la solution, il faudra taper la commande
draw_sudoku(solution, "Solution")$
en dessous du Sudoku obtenu.
Le calcul par Maxima peut prendre quelques minutes (5 minutes sur ma machine), dans la mesure ou l'algorithme de génération est assez lent. Le recalcul complet du notebook génère un nouveau Sudoku.
L'IA Claude a aidé pour la mise au point de ce notebook proposé en version 1.0.

```
--> kill(all)$
    set_random_state(make_random_state(true))$

    /* ---------- Outils de base ---------- */

    shuffle_list(L) := block(
      [A : copylist(L), i, j, tmp, n : length(L)],
      for i : n step -1 thru 2 do (
         j : random(i) + 1,
         tmp : A[i], A[i] : A[j], A[j] : tmp
      ),
      return(A)
    )$

    copy_grid(Grid) := makelist(copylist(Grid[i]), i, 1, 9)$

    /* Remplacement d'une cellule sans effet de bord */
    set_cell(Grid, r, c, val) := makelist(
      if i = r then makelist(if j = c then val else Grid[i][j], j, 1, 9)
      else Grid[i],
      i, 1, 9
    )$

    row_values(Grid, r) := Grid[r]$
    col_values(Grid, c) := makelist(Grid[r][c], r, 1, 9)$

    box_values(Grid, r, c) := block(
      [r0, c0, vals : [], i, j],
      r0 : 3·floor((r-1)/3) + 1, c0 : 3·floor((c-1)/3) + 1,
      for i : r0 thru r0+2 do
         for j : c0 thru c0+2 do vals : endcons(Grid[i][j], vals),
      return(vals)
    )$

    /* Test de présence */
    member_p(x, L) := block([found : false, k],
      for k : 1 thru length(L) do if L[k] = x then found : true,
      return(found)
    )$

    allowed_values(Grid, r, c) := block([used, d, res : []],
      if Grid[r][c] # 0 then return([]),
      used : append(row_values(Grid,r), col_values(Grid,c), box_values(Grid,r,c)),
      for d : 1 thru 9 do
         if not member_p(d, used) then res : endcons(d, res),
```

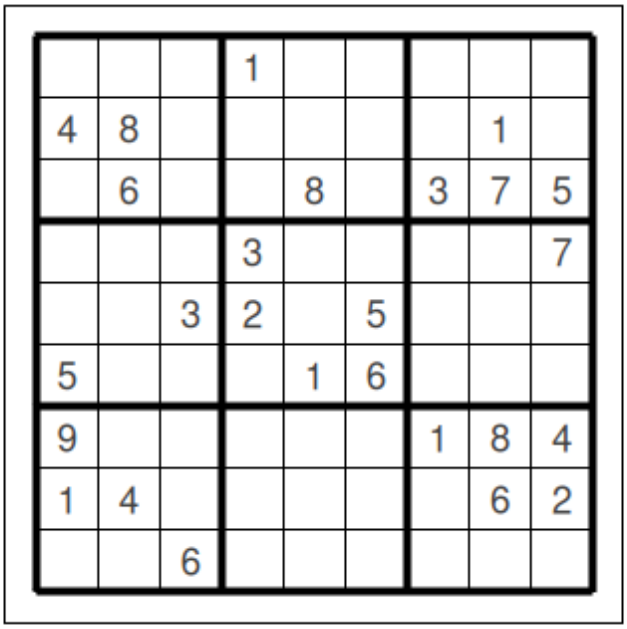
```
  return(res)
)$

find_best_empty_cell(Grid) := block(
  [r, c, best : false, best_len : 10, cand, l],
  for r : 1 thru 9 do
    for c : 1 thru 9 do
      if Grid[r][c] = 0 then (
        cand : allowed_values(Grid, r, c), l : length(cand),
        if l < best_len then (best : [r, c], best_len : l)
      ),
  return(best)
)$

/* ---------- Génération d'une grille complète ---------- */

base_pattern(r, c) := mod((c−1) + 3·(r−1) + floor((r−1)/3), 9) + 1$

generate_solution() := block(
  [Grid, row_perm, col_perm, num_perm, r, c],
  Grid : makelist(makelist(base_pattern(r, c), c, 1, 9), r, 1, 9),

  num_perm : shuffle_list([1,2,3,4,5,6,7,8,9]),
  Grid : makelist(makelist(num_perm[Grid[r][c]], c, 1, 9), r, 1, 9),

  row_perm : append(shuffle_list([1,2,3]), shuffle_list([4,5,6]), shuffle_list([7,8,9])),
  Grid : makelist(Grid[row_perm[r]], r, 1, 9),

  col_perm : append(shuffle_list([1,2,3]), shuffle_list([4,5,6]), shuffle_list([7,8,9])),
  Grid : makelist(makelist(Grid[r][col_perm[c]], c, 1, 9), r, 1, 9),

  return(Grid)
)$

/* ---------- Comptage récursif de solutions ---------- */

count_solutions_rec(Grid, count, lim) := block(
  [pos, r, c, cands, i, res, grid_next],
  if count >= lim then return(count),
  pos : find_best_empty_cell(Grid),
  if pos = false then return(count + 1),

  r : pos[1], c : pos[2],
  cands : allowed_values(Grid, r, c),
  if length(cands) = 0 then return(count),

  res : count,
  for i : 1 thru length(cands) do (
    if res < lim then (
      grid_next : set_cell(Grid, r, c, cands[i]),
      res : count_solutions_rec(grid_next, res, lim)
    )
  ),
  return(res)
)$

unique_solution_p(Grid) := is(count_solutions_rec(Grid, 0, 2) = 1)$
```

```
/* ---------- Suppression de cases ---------- */

all_positions() := block([P : [], r, c],
  for r : 1 thru 9 do for c : 1 thru 9 do P : endcons([r,c], P),
  return(P)
)$

remove_cells_unique(Grid) := block(
  [P, k, r, c, nb_retires : 0, grid_try],
  P : shuffle_list(all_positions()),
  for k : 1 thru length(P) do (
    r : P[k][1], c : P[k][2],
    if Grid[r][c] # 0 then (
      grid_try : set_cell(Grid, r, c, 0),
      if unique_solution_p(grid_try) then (
        Grid : grid_try,
        nb_retires : nb_retires + 1
      )
    )
  ),
  return(Grid)
)$
/* la variable nb_retires donne le nombre de cases retirées de la grille de départ */

/* ---------- Affichage ---------- */

grid_to_matrix(Grid) := apply('matrix, Grid)$
solution : generate_solution()$
sudoku : copy_grid(solution)$
sudoku : remove_cells_unique(sudoku)$

draw_sudoku(Grid, titre) := block(
  [segs_fins : [], segs_epais : [], labels_list : [], r, c, val, xg, yg],

  for i : 0 thru 9 do (
    if mod(i, 3) = 0 then (
      segs_epais : endcons([2·i, 0, 2·i, 18], segs_epais),
      segs_epais : endcons([0, 2·i, 18, 2·i], segs_epais)
    ) else (
      segs_fins : endcons([2·i, 0, 2·i, 18], segs_fins),
      segs_fins : endcons([0, 2·i, 18, 2·i], segs_fins)
    )
  ),

  for r : 1 thru 9 do
    for c : 1 thru 9 do (
      val : Grid[r][c],
      if val # 0 then (
        xg : 2·(c−1) + 1,
        yg : 2·(9−r) + 1,
        labels_list : endcons(label([string(val), xg, yg]), labels_list)
      )
    ),

  wxdraw2d(
    line_width = 1,
    color = black,
    points_joined = true,
```

```
        point_type = none,
        apply(append, map(lambda([s], [points([s[1],s[3]],[s[2],s[4]])]), segs_fins)),

        line_width = 4,
        color = black,
        points_joined = true,
        point_type = none,
        apply(append, map(lambda([s], [points([s[1],s[3]],[s[2],s[4]])]), segs_epais)),

        font_size = 14,
        font = "Helvetica",
        color = "#404040",
        labels_list,

        xrange = [−1, 19],
        yrange = [−1, 19],
        proportional_axes = xy,
        grid = false,
        xaxis = false,
        yaxis = false,
        xtics = false,
        ytics = false
      )
    )$
```

```
--> draw_sudoku(sudoku, "Sudoku")$
```

(%t30)



Created with wxMaxima.