70083 ExerciseTypes.CW1

C++ 1

Submitters

sf23 Shihan Fu

76/100

Good job!

Emarking

Sudoku TestSummary.txt: 1/1 Shihan Fu - sf23:v5

```
1: Detailed Output for test: Task 1
3:
4: Task 1
5:
6: Compiled OK
7 •
8: Compilation Standard Output:
10: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
12: Test Passed
13:
14:
15:
17: Detailed Output for test: Task 2
19:
20: Task 2
21:
22: Compiled OK
23:
24: Compilation Standard Output:
26: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
28: Test failed because Output differs
29:
30: Model Output (Left) vs Student's Output (Right):
34: Loading Sudoku board from file 'easy.dat' ... Success!
35: Putting '1' into I8 is a valid move. The board is:
36: 1 2 3 4 5 6 7 8 9
37: +======+====+
38: A | : : | 1 : : 8 | 3 : : |
40: B | 2: 4: | :5: | : : |
42: C | : : 8 | : : | : 6:1 |
44: D | : : 4 | : : 9 | : : 3 |
46: E | : 6: | : 2: |
48: F | 3: : | 8: : | 1: : |
49: +======+
50: G | 1:7: | : : | 9: : |
52: H | : : | :1: | :5:2|
54: I | : : 2 | 7 : : 4 | : 1 : |
55: +======+====+
56: Loading Sudoku board from file 'easy.dat'... Success!
58: Putting '3' into F8 is NOT a valid move. The board is:
59: 1 2 3 4 5 6 7 8 9
61: A | : : | 1 : : 8 | 3 : : |
```

```
Loading Sudoku board from file 'easy.dat' ... Success!
Putting '1' into I8 is a valid move. The board is:
  1 2 3 4 5 6 7 8 9
 +=====++===++===++
A | : : | 1 : : 8 | 3 : : |
B | 2 : 4 : | : 5 : | : : |
C | : :8 | : : | :6:1 |
D | : : 4 | : : 9 | : : 3 |
E | :6: | :2: |
F | 3: : | 8: : | 1: : |
 +=====+===+
G | 1:7: | : : | 9: : |
H | : : | :1: | :5:2 |
 +---+---+
I | : : 2 | 7 : : 4 | : 1 : |
 Loading Sudoku board from file 'easy.dat'... Success!
Putting '3' into F8 is NOT a valid move. The board is:
 1 2 3 4 5 6 7 8 9
A | : : | 1 : : 8 | 3 : : |
```

```
62: +---+---+
63: B | 2:4: | :5: | : : |
64: +---+---+
65: C | : : 8 | : : | : 6 : 1 |
66: +======+====+
67: D | : : 4 | : : 9 | : : 3 |
68: +---+---+
69: E | : 6: | : 2: |
70: +---+---+
71: F | 3 : : | 8 : : | 1 : : |
72: +======+====+
73: G | 1 : 7 : | : : | 9 : : |
74: +---+--+
75: H | : : | : 1 : | : 5 : 2 |
76: +---+---+
77: I | : : 2 | 7 : : 4 | : : |
78: +======+====+
79: Loading Sudoku board from file 'easy.dat' ... Success!
81: Putting '3' into B5 is NOT a valid move. The board is:
82: 1 2 3 4 5 6 7 8 9
83: +======+====+
84: A | : : | 1 : : 8 | 3 : : |
85: +---+---+
86: B | 2 : 4 : | : 5 : | : : |
88: C | : : 8 | : : | : 6 : 1 |
90: D | : : 4 | : : 9 | : : 3 |
92: E | : 6 : | : 2 : |
94: F | 3: : | 8: : | 1: : |
96: G | 1:7: | : : | 9: : |
98: H | : : | : 1 : | : 5 : 2 |
100: I | : : 2 | 7 : : 4 | : : |
101: +======+====+
102: Loading Sudoku board from file 'easy.dat' ... Success!
104: Putting '0' into E5 is NOT a valid move. The board is:
105: 1 2 3 4 5 6 7 8 9
106: +======+====+
107: A | : : | 1 : : 8 | 3 : : |
109: B | 2 : 4 : | : 5 : | : : |
110: +---+---+
111: C | : : 8 | : : | : 6 : 1 |
112: +======+====+
113: D | : : 4 | : : 9 | : : 3 |
114: +---+---+
115: E | : 6: | : 2: |
116: +---+---+
117: F | 3 : : | 8 : : | 1 : : |
118: +======+====+
119: G | 1 : 7 : | : : | 9 : : |
120: +---+---+
121: H | : : | : 1 : | : 5 : 2 |
122: +---+---+
```

Sudoku

В	2	: 4		:	: 5	:		:	:	
С		:	+ : 8	:	:	:		: 6	: 1	
D	İ	:	====- : 4 +	:	:	: 9	İ	:	: 3	
E		: 6	: :	:	:	:		: 2		
F	3	:	: :	8 :	:	:	1	:	:	
G	1	: 7	:	:	:	:	9	:	:	
Н		:		:	: 1	:		: 5	: 2	
		:	: 2	7 :	:	: 4		:	:	
			====+ doku							t' Success!
Put										ne board is:
-			3 ====+	·====		====	+===		====	+
Α.	1	: +	: +			: 8 +				
В		: 4	:		: 3			: +	: +	
С		:	: 8 ====	:	:	:		: 6		
D		:	: 4	:	:	: 9		:	: 3	
E		: 6	:	:	:	:		: 2	:	 -
F	3	:	: =====	8 :	:	:	1	:	:	
G	1	: 7		:	:	:	9	:	:	
Н		: :	:	:	: 1	:	ĺ	: 5	: 2 +	
I		:	: 2 ====	7 :	:	: 4		:	:	
										at' Success!
	1	2	3	4	5	6	7	8	9	e. The board is:
A		:		1 :	:	: 8	3	:	:	+
В	2		:	:	: 5			:	+ :	†
C		:	: 8	:	:	:		: 6	: 1	†
D	İ	:	: 4	:	:	: 9	İ	:	: 3	
E		: 6		:	:	:		: 2	:	
F	3	:	:	8 :	:	:	1	:	:	
G	1	: 7		:	:	:	9	:	====- :	!
Н				:	: 1	:		: 5	+ : 2	+
-	+	+	++	++		+	+	+	+	+

```
123: I | : : 2 | 7 : : 4 | : : |
124: +======+====+
125: Loading Sudoku board from file 'easy.dat' ... Success!
127: Putting 'Z' into E5 is NOT a valid move. The board is:
128: 1 2 3 4 5 6 7 8 9
129: +======+====+
130: A | : : | 1 : : 8 | 3 : : |
131: +---+---+
132: B | 2 : 4 : | : 5 : | : : |
133: +---+---+
134: C | : : 8 | : : | : 6:1 |
135: +======+=====+
136: D | : : 4 | : : 9 | : : 3 |
137: +---+---+
138: E | : 6 : | : 2 : |
139: +---+---+
140: F | 3 : : | 8 : : | 1 : : |
141: +======+====+
142: G | 1 : 7 : | : : | 9 : : |
143: +---+---+
144: H | : : | : 1 : | : 5 : 2 |
146: I | : : 2 | 7 : : 4 | : : |
147: +======+====+
148: Loading Sudoku board from file 'easy.dat' ... Success!
149: Putting '8' into II is NOT a valid move. The board is:
150: 1 2 3 4 5 6 7 8 9
151: +======+====+
152: A | : : | 1 : : 8 | 3 : : |
153: +---+---+
154: B | 2 : 4 : | : 5 : | : : |
155: +---+---+
156: C | : : 8 | : : | : 6 : 1 |
157: +======+====+
158: D | : : 4 | : : 9 | : : 3 |
159: +---+---+
160: E | : 6: | : 2: |
161: +---+---+
162: F | 3 : : | 8 : : | 1 : : |
163: +======+====+
164: G | 1 : 7 : | : : | 9 : : |
165: +---+---+
166: H | : : | : 1 : | : 5 : 2 |
168: I | : : 2 | 7 : : 4 | : : |
169: +======+====+
170: Loading Sudoku board from file 'easy.dat' ... Success!
171: Putting '5' into B0 is NOT a valid move. The board is:
172: 1 2 3 4 5 6 7 8 9
173: +======+====+
174: A | : : | 1 : : 8 | 3 : : |
175: +---+---+
176: B | 2 : 4 : | : 5 : | : : |
177: +---+---+
178: C | : : 8 | : : | : 6 : 1 |
179. +======+=====+
180: D | : : 4 | : : 9 | : : 3 |
181: +---+---+
182: E | : 6 : | : 2 : |
183: +---+---+
```

xτ	: 3	/	2										
							: 4			:			
												Success!	
	1	2	2	3	4	5	6	7	8			board is:	
A		:	:		1 :	:	: 8 +	3	:	:			
в	2	: 4	1:		:	: 5	:		:	:			
c		:	:	8	:	:			: 6	: 1			
D		:	:	4	:	:	: 9		:	: 3			
E		: 6	ŝ :		:	:	: :		: 2				
F	3	:	:		8 :	:	:	1	:				
G	1	: 7	7 :		:	:	:	9	:	:			
н		:	:		:	: 1	:		: 5	: 2			
		:	:	2	7 :	:	: 4		:	:			
Loa	adino tino	g 9 g 1	8ud 8'	loku int 3	boar to II	rd f I is 5	rom : NOT 6	file a v 7	ea alid 8	sy.da move	at' e. The	Success! board is:	
A		:	:		1 :	:	: 8	3	:				
в		: 4	1:		;	: 5	: :		:	:			
C		:	:	8	;	:	:		: 6	: 1			
D		:	:	4	:	:	: 9		:				
E		: 6	:		:	:	+ : +		: 2	:			
F	3	:	:		8 :	:	:	1	:	:			
G	1	: 7	7 :		:	:	:	9	:	:			
н		:	:		;	: 1	:		: 5	: 2			
I		:	:	2	7 :	:	: 4		:	: :====+			
Loa Put	adino tino	g \$	Sud 5'	loku int 3	boar to B(rd f) is	rom : NOT 6	file a v 7	ea alid 8	sy.da move	at' e. The	Success! board is:	
A		:	:		1 :	:	: 8	3	:				
	2	: 4	1:		:	: 5	:		:	:			
c		:	:	8	:	:	:		: 6	: 1			
D		:	:	4	:	:	: 9		:	: 3			
E		: 6	ŝ :		 :	:			: 2				
7		,	-+						,	,	1		

```
184: F | 3 : : | 8 : : | 1 : : |
185: +======+====+
186: G | 1 : 7 : | : : | 9 : : |
187: +---+---+
188: H | : : | :1: | :5:2|
189: +---+---+
190: I | : : 2 | 7 : : 4 | : : |
191: +=====+
192:
194: Detailed Output for test: Task 3
195: -----
196:
197: Task 3
198:
199: Compiled OK
200:
201: Compilation Standard Output:
202:
203: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
204:
205: Test Passed
206:
207:
208:
209:
210: Detailed Output for test: Task 4
211: -----
212:
213: Task 4
214:
215: Compiled OK
216:
217: Compilation Standard Output:
218:
219: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
221: Test Passed
222:
223:
226: Detailed Output for test: Task 5
228:
229: Task 5
230:
231: Compiled OK
232:
233: Compilation Standard Output:
234:
235: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
236:
237: Test Passed
238:
239:
240:
241:
242: Detailed Output for test: Additional puzzles
243: -----
```

Sudoku

^	•		-	, –													
												1 +===					
G	İ	1	:	7	:		İ	:		:		9	:		:		
	İ		:		:		İ	:	1	:		ļ 	:	5	:	2	:
Ι	İ		:		:	2	7	:		:	4	 -===	:		:		

```
245: Additional puzzles
246:
247: Compiled OK
248:
249: Compilation Standard Output:
250:
251: g++ -Wall -g main.cpp sudoku.cpp -o sudoku
252:
253: Test Passed
254:
255:
```

sudoku.h: 1/1

Nice to see some function comments, but more detail would be better!

```
Sudoku
   1: #include "sudoku.h"
   2: #include <cassert>
   3: #include <cstdio>
   4: #include <cstring>
   5: #include <fstream>
    6: #include <iostream>
   8: using namespace std;
  10: /* You are pre-supplied with the functions below. Add your own
  11: function definitions to the end of this file. */
  13: /* pre-supplied function to load a Sudoku board from a file */
  14: void load_board(const char *filename, char board[9][9]) {
  15:
  16: cout << "Loading Sudoku board from file '" << filename << "'... ";
  17:
  18: ifstream in(filename);
  19: if (!in) {
  20:
        cout << "Failed!\n";
  21: }
  22: assert(in);
  23:
  24: char buffer[512];
  25:
  26: int row = 0;
  27: in.getline(buffer, 512);
  28: while (in && row < 9) {
  29: for (int n = 0; n < 9; n++) {
   30:
          assert(buffer[n] == '.' | isdigit(buffer[n]));
  31:
           board[row][n] = buffer[n];
  32:
  33: row++:
  34:
         in.getline(buffer, 512);
  35: }
  36:
  37: cout << ((row == 9) ? "Success!" : "Failed!") << '\n';
  38: assert (row == 9);
  39: }
   40:
   41: /* internal helper function */
   42: void print_frame(int row) {
  43: if (!(row % 3)) {
  44: cout << " +======++====++n":
  45: } else {
        cout << " +---+--+\n";
  47: }
  48: }
  49:
   50: /* internal helper function */
   51: void print_row(const char *data, int row) {
   52: cout << (char) ('A' + row) << " ";
  53: for (int i = 0; i < 9; i++) {
  54: cout << ((i % 3) ? ':' : ' | ') << " ";
  55: cout << ((data[i] == '.') ? ' ' : data[i]) << " ";
  56: }
   57: cout << "\\n";
  58: }
  59 .
   60: /* pre-supplied function to display a Sudoku board */
   61: void display_board(const char board[9][9]) {
   62: cout << " ";
   63: for (int r = 0; r < 9; r++) {
        cout << (char) ('1' + r) << " ";
   64:
   65: }
   66: cout << '\n';
```

```
Sudoku
                                     sudoku.cpp: 2/3
                                                                      Shihan Fu - sf23:v5
   67: for (int r = 0; r < 9; r++) {
        print frame(r);
         print row(board[r], r);
   70: }
   71: print_frame(9);
   72: }
   73:
   74: /* my own functions */
   75:
   76: /* check if it is valid to put digit in a given position */
   77: bool make_move(string position, char digit, char board[9][9]) {
   78: // check input position's length
   79: if (position.length() > 3) {
        return false;
  81: }
   82: // check position's content
   83: if (position[0] >= 'A' \&\& position[0] <= 'I' \&\& position[1] >= '1' \&\&
   84.
            position[1] <= '9') {
                                       This should be restructured to
   85:
          // the position is good
                                       not have an empty if block.
   86:
        } else {
                                       This is not very elegant.
   87:
          return false;
   88: }
   89:
   90: // check digit
                                                                    17/25 Pretty good
   91: if (digit < '1' | digit > '9') {
   92 .
         return false;
   93: }
  94:
   95:
        // check logic
        int row = position[0] - 65; Please use ASCII literals in times like this!
        int col = position[1] - 48 - 1;
   97:
  98:
        for (int i = 0; i < 9; i++) {
  99.
  100:
          if (i != col && board[row][i] == digit) {
  101:
            // row check fail
  102:
            return false:
                                         -4 Did not check if digit appears in the current 3x3 block
  103:
  104:
  105:
          if (i != row && board[i][col] == digit) {
  106:
          // column check fail
  107:
            return false;
  108:
  109: }
  110:
  111: // update the board
        board[row][col] = digit;

    -4 Did not check if position is already filled

  113: return true;
  114: }
  115.
  116: /* checks whether all board positions are occupied by digits, and false
  117: * otherwise. No logical check*/
  118: bool is_complete(const char board[9][9]) {
  119: for (int i = 0; i < 9; i++) {
          for (int j = 0; j < 9; j++) {
  120:
  121:
            // if is blank, will be replaced by a '.'
  122:
            if (board[i][j] == '.') {
  123:
              return false; You don't actually need this first check.
  124:
  125:
            // if is not a digit between 1 to 9
  126:
            if (board[i][j] < '1' | board[i][j] > '9') {
  127:
              return false;
  128:
  129:
          }
                                       10/10 Looks good
```

```
Sudoku
                                      sudoku.cpp: 3/3
                                                                        Shihan Fu - sf23:v5
  134: /* save the data in a file */
  135: bool save board(string filename, char board[9][9]) {
  136: ofstream output;
  137: output.open(filename);
  138: if (!output) {
  139.
          // fali to open file Good to check if file opened correctly...
  140:
           return false;
  141:
  142: for (int i = 0; i < 9; i++) {
  143.
           for (int j = 0; j < 9; j++) {
                                                                 12/15 Good job
  144:
            // put chars into the output stream
  145:
             output.put(board[i][j]);
  146:
  147:
           output.put('\n');
  148: }
 149: output.close();-3 But also need to check if anything went wrong while writing!
  151: }
  152:
  153: /* solve the sudoku puzzle with recursions */
  154: bool solve_board(char board[9][9]) {
                                               Putting all the logic inside the nested loops here
  155: for (int i = 0; i < 9; i++) {
                                               does work, given your return statements in the if
           for (int j = 0; j < 9; j++) {
                                               statement, but it isn't particularly neat.
  157:
             // find am empty cell
  158:
             if (board[i][j] == '.') {
  159:
               // try to put in a number from 1 to 9
  160:
               for (char number = '1'; number <= '9'; number++) {</pre>
  161:
                 if (make move index(i, j, number, board)) {
                   // if valid operation
  162:
  163:
                   board[i][j] = number;
  164:
                   if (solve_board(board))
  165:
                     // recur the board
  166:
                     return true;
  167:
  168:
                   // if put in this number cannot lead to a valid solution, replace it
  1.69:
                   board[i][j] = '.';
  170:
                                                       24/30 This is very close to working, but the
  171:
                                                      make_move_index logic is not quite correct.
  172:
               // if no valid operation, trace back
  173:
               return false:
  174:
  175:
  176: }
  177: // if there is no empty cells, find a solution
  178: return true;
  179: }
  180 •
  181: /* check if it is valid to put digit in a given position with index and board
  182: * area check*/
  183: bool make move index(int row, int col, char digit, char board[9][9]) {
  184: for (int i = 0; i < 9; i++) {
  185:
          // if the digit appeared in the tor ow column or the block area, the move
  186:
           // cannot be successful
  187:
           if (board[row][i] == digit | | board[i][col] == digit | |
  188:
               board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == digit) {
  189:
             return false;
  190:
                              This logic is not actually checking the 3x3 block correctly -
  191: }
                              it is only checking the diagonal of the block.
  192: return true;
  193: }
```

130: 131:

132: }

return true;

```
Shihan Fu - sf23:v5
Sudoku
                                 main.cpp: 1/3
                                                                                     Sudoku
                                                                                                                       main.cpp: 2/3
                                                                                                                                                   Shihan Fu - sf23:v5
   1: #include <iostream>
                                                                                        67: // cout << "a valid move. The board is:\n";
   2: #include <cstdio>
                                                                                        68: // display board(board);
   3: #include<cstring>
   4: #include<vector>
                                                                                        70: // // test 3
   5: #include <fstream>
                                                                                        71: // cout << "Putting '9' into Z5 is ";
   6: #include <cassert>
   7: #include <time.h>
                                                                                        73: // if (!make_move("Z5", '9', board)) {
   8: #include "sudoku.h"
                                                                                        74: // cout << "NOT ";
                                                                                        75: // }
                                                                                        76: // cout << "a valid move. The board is:\n";
  10: using namespace std;
                                                                                        77: // display_board(board);
  11:
  12: int main() {
                                                                                        78:
  13:
                                                                                        79:
  14: char board[9][9];
                                                                                        80:
  15:
                                                                                        81 •
  16:
                                                                                        82:
  17:
                                                                                        18: /* This section illustrates the use of the pre-supplied helper functions. */
                                                                                        84:
  19: // cout << "====== Pre-supplied functions ======= \n\n";
                                                                                        85: // load board("easy.dat", board);
  20:
                                                                                        86: // if (save board("easy-copy.dat", board)) {
  21: // cout << "Calling load board():\n";
                                                                                        87: // cout << "Save board to 'easy-copy.dat' successful.\n";
                                                                                        88: // } else {
  22: // load_board("easy.dat", board);
  23:
                                                                                        89: // cout << "Save board failed.\n";
                                                                                        90: // }
  24: // cout << '\n';
  25: // cout << "Displaying Sudoku board with display_board():\n";
                                                                                        91: // cout << '\n';
  26: // display_board(board);
                                                                                        92:
  27: // cout << "Done!\n\n";
                                                                                        93: // cout << "=============\n\n";
  28:
                                                                                        94:
  29:
                                                                                        95:
  30: // cout << "========\n\n";
                                                                                        96: // load board("easy.dat", board);
  31:
                                                                                        97:
  32: // load_board("easy.dat", board);
                                                                                        98: // if (solve_board(board)) {
  33: // cout << "Board is ";
                                                                                        99: // cout << "The 'easy' board has a solution:\n";
                                                                                       100: // display_board(board);
  34: // if (!is_complete(board)) {
  35: // cout << "NOT ";
                                                                                       101: // } else {
  36: // }
                                                                                       102: // cout << "A solution cannot be found.\n";
                                                                                       103: // }
  37:  // cout << "complete.\n\n";</pre>
  38 .
                                                                                       104: // cout << '\n';
  39: // load_board("easy-solution.dat", board);
                                                                                       105:
  40: // cout << "Board is ";
                                                                                       106: // load_board("medium.dat", board);
                                                                                       41: // if (!is complete(board)) {
                                                                                       108: // cout << "The 'medium' board has a solution:\n";
  42: // cout << "NOT ";
                                                                                       109: // display_board(board);
  43: // }
  44: // cout << "complete.\n\n";
                                                                                       110: // } else {
  45:
                                                                                       111: // cout << "A solution cannot be found. \n";
  46: // cout << "========= Ouestion 2 ========\n\n";
                                                                                       112: // }
  47:
                                                                                       113: // cout << '\n';
  48: // load_board("easy.dat", board);
                                                                                       114:
  49 •
                                                                                       115: // write more tests
  50: // // test 1
                                                                                       116: // load_board("mystery2.dat", board);
  51: // cout << "Putting '1' into I8 is ";
                                                                                       118: // if (solve_board(board)) {
                                                                                       119: // cout << "The 'mystery2' board has a solution:\n";
120: // display_board(board);</pre>
  53: // if (!make_move("I8", '1', board)) {
  54: // cout << "NOT ";
                                                                                       121: // } else {
  55: // }
                                                                                       122: // cout << "A solution cannot be found.\n";
  56: // cout << "a valid move. The board is:\n";
  57: // display_board(board);
                                                                                       123: // cout << "This is the origin board.\n";
  58:
                                                                                       124: // load_board("mystery2.dat", board);
  59: // //write more tests
                                                                                       125: // display_board(board);
  60:
                                                                                       126:
  61: // // test 2
                                                                                       127: // }
  62: // cout << "Putting '0' into I8 is ";
                                                                                       128:
                                                                                       129: // cout << '\n';
  64: // if (!make_move("I8", '0', board)) {
                                                                                       130:
  65: // cout << "NOT ";
                                                                                       131: // cout << "=======\n\n";
  66: // }
```

```
133: // // write more tests
134: clock t tStart = clock();
136: load_board("mystery2.dat", board);
137:
138: for(int i=0;i<100;i++){
139: solve_board(board);
140: }
141:
142: clock_t tEnd = clock();
143: if(solve_board(board)){
144: cout << "The 'mystery2' board has a solution:\n";
145:
      display_board(board);
146: } else {
147: cout << "A solution cannot be found.\n";
148: }
149:
150: cout << '\n';
151:
152:
      cout << "The time used is : " << tEnd - tStart << endl;</pre>
153:
154:
155:
156: return 0;
157: }
158:
159:
160:
161:
```

162:

Sudoku findings.txt: 1/1 Shihan Fu - sf23:v5

```
1: Summarise your findings here (see specification).
   2: 05 Before the validation, I define how to judge whether a puzzle is "hard". We /
think the more recursions one puzzle needs, the more difficult it is.
   3: If a puzzle needs more recursions to find a solution, it needs more time to /
execute. So I used the "time.h" to help me calculate the running time.
   4: I set two variables to record the time when the function of solve_board is /
started and it ended and substracting them to get the time duration of executing.
   5: In order to minimize the random CPU error, I used for loops to execute the /
function for multiple times. (100 in this case)
   6:
   7:
        program result:
   8:
        mysterv1.dat ->SUCESS 92995
        mystery2.dat ->CANNOT BE FOUND
  10:
         mystery3.dat ->SUCESS 3172
  11:
  12:
         conclusion: according to the time used to solve these puzzles,
  13:
         mysteryl is extremely hard
  14:
             mystery2 is impossible to solve
  15:
           mystery3 is hard
```

You have correctly identified the boards, but I would like to see some more analysis here!

Why is your method of averaging the times a good measure of difficulty? What are you actually measuring? Is it the efficiency of your solve_board function?

What makes one board more difficult to solve than another? Does the way your algorithm is written affect that?

Also, whilst averaging the timings is a reasonable method, I would have liked to have seen some recursion call counting.

13/20