

UTSGamesstudio / cs280-assb-2017-chehtien

Code

Issues

Pull requests

Actions

Projects

Security

Insights

master ▾

...

cs280-assb-2017-chehtien / Sudoku.cpp



chehtien mostly done



1 contributor

Raw

Blame



194 lines (155 sloc) | 5.4 KB

```
1  // #define DEBUG_VALID
2
3  #include "Sudoku.h"
4  #include <iostream>
5  #include <memory>
6
7  using std::cout;
8  using std::endl;
9
10 Sudoku::Sudoku(int basesize, SymbolType stype, CALLBACK callback): moves_{0}, stype_{stype}, ca
11     stats_.basesize = basesize;
12     board_len_ = basesize * basesize;
13     board_ = new char[stats_.basesize];
14 }
15
16 Sudoku::~Sudoku() {
17     delete[] board_;
18 }
19
20 void Sudoku::SetupBoard(const char *values, size_t size) {
21     // fill board with values
22     for (size_t i=0; i<size; ++i) {
23         if (values[i] == '.')
24             board_[i] = ' ';
25         else
26             board_[i] = values[i];
27     }
28 }
29
30 bool Sudoku::Solve() {
```

```
31     unsigned x = 0;
32     unsigned y = 0;
33
34     callback_(*this, board_, MessageType::MSG_STARTING, moves_, stats_.basesize, -1, 0);
35
36     auto success = place_value(x, y);
37     if (success)
38         callback_(*this, board_, MessageType::MSG_FINISHED_OK, moves_, stats_.basesize, -1, 0);
39     else
40         callback_(*this, board_, MessageType::MSG_FINISHED_FAIL, moves_, stats_.basesize, -1, 0);
41
42     return success;
43 }
44
45 bool Sudoku::place_value(unsigned x, unsigned y) {
46     // base case is when reach 1 after last row
47     if (y == board_len_)
48         return true;
49
50     // get the 1d index into the board
51     unsigned index = x + board_len_ * y;
52
53     #ifdef DEBUG_VALID
54         cout << "place_value: starting placement of (" << x << ", " << y << ") ";
55     #endif
56
57     // check if pos already occupied
58     if (board_[index] != ' ') {
59
60         #ifdef DEBUG_VALID
61             cout << "NOT PLACED (OCCUPIED), going next pos" << endl;
62         #endif
63
64         // recurse to next position
65         if (x == board_len_ - 1) {
66             if (place_value(0, y+1))
67                 return true;
68         }
69         else {
70             if (place_value(x+1, y))
71                 return true;
72         }
73         return false;
74     }
75
76     // init the correct type of val
77     char val;
78     if (stype_ == SymbolType::SYM_NUMBER)
79         val = '1';
80     else
81         val = 'A';
82 }
```

```

83 // loop thru all possible vals and attempt to place them
84 for (size_t i=0; i<board_len_; ++i) {
85     // check if driver called abort
86     if (callback_(*this, board_, MessageType::MSG_ABORT_CHECK, moves_, stats_.basesize, i)
87
88 #ifdef DEBUG_VALID
89     cout << "ABORTED by user" << endl;
90 #endif
91
92     return false;
93 }
94
95 // attempt to place val
96 board_[index] = val;
97 stats_.moves = ++moves_;
98 ++stats_.placed;
99 callback_(*this, board_, MessageType::MSG_PLACING, moves_, stats_.basesize, index, val);
100 if (is_valid(x, y, val)) {
101
102 #ifdef DEBUG_VALID
103     cout << "VALID POS and PLACING " << val << endl;
104 #endif
105
106     // recurse to next position
107     if (x == board_len_ - 1) {
108         if (place_value(0, y+1)) {
109             return true;
110         }
111     }
112     else {
113         if (place_value(x+1, y)) {
114             return true;
115         }
116     }
117
118     // all vals exhausted so backtrack
119     board_[index] = ' ';
120     //--stats_.placed;
121     ++stats_.backtracks;
122     callback_(*this, board_, MessageType::MSG_REMOVING, moves_, stats_.basesize, index,
123 }
124
125 // next val
126 board_[index] = ' ';
127 --stats_.placed;
128 callback_(*this, board_, MessageType::MSG_REMOVING, moves_, stats_.basesize, index, val);
129 ++val;
130 }
131
132 return false;
133 }
134

```

```

135 bool Sudoku::is_valid(unsigned x, unsigned y, char val) {
136     unsigned index = x + board_len_ * y;
137
138     #ifdef DEBUG_VALID
139         cout << "is_valid: validating insert of " << val << " in (" << x << "," << y << ") ";
140     #endif
141
142     // check if same values in row and col
143     for (size_t i=0; i<board_len_; ++i) {
144
145         // #ifdef DEBUG_VALID
146             // cout << "is_valid: comparing val " << val << " with (" << x << "," << i << ")=" << board[i] << endl;
147         // #endif
148         unsigned next_col_slot_index = i + board_len_ * y;
149         unsigned next_row_slot_index = x + board_len_ * i;
150         if ( ((next_row_slot_index != index) && (board[next_row_slot_index] == val))
151             || ((next_col_slot_index != index) && (board[next_col_slot_index] == val))) {
152
153             #ifdef DEBUG_VALID
154                 cout << " INVALID (EXISTS ON ROW/COL)" << endl;
155             #endif
156
157             return false;
158         }
159     }
160
161     // check remaining 4 values in the sector
162     size_t sectorrow = stats_.basesize * (y/stats_.basesize);
163     size_t sectorcol = stats_.basesize * (x/stats_.basesize);
164     size_t row1 = (y + stats_.basesize - 1) % stats_.basesize;
165     size_t row2 = (y + stats_.basesize + 1) % stats_.basesize;
166     size_t col1 = (x + stats_.basesize - 1) % stats_.basesize;
167     size_t col2 = (x + stats_.basesize + 1) % stats_.basesize;
168
169     #ifdef DEBUG_VALID
170         // cout << " testing board[" << index << " ]=" << board[index] << endl;
171     #endif
172
173     if ( (board[(col1+sectorcol) + board_len_ * (row1+sectorrow)] == val)
174         || (board[(col1+sectorcol) + board_len_ * (row2+sectorrow)] == val)
175         || (board[(col2+sectorcol) + board_len_ * (row1+sectorrow)] == val)
176         || (board[(col2+sectorcol) + board_len_ * (row2+sectorrow)] == val) ) {
177
178         #ifdef DEBUG_VALID
179             cout << " INVALID (EXISTS IN SECTOR)" << endl;
180         #endif
181
182         return false;
183     }
184
185     return true;
186 }

```

```
187
188     const char* Sudoku::GetBoard() const {
189         return board_;
190     }
191
192     Sudoku::SudokuStats Sudoku::GetStats() const {
193         return stats_;
194     }
```