UTSGamesstudio / cs280-assb-2017-chektien

Code Issues Pull requests Actions Projects Security Insights

master

cs280-assb-2017-chektien / Sudoku.cpp

chektien mostly done

1 contributor

Raw Blame

194 lines (155 sloc) | 5.4 KB

```
//#define DEBUG_VALID

#include "Sudoku.h"
#include <iostream>
#include <memory>

using std::cout;
using std::endl;

Sudoku::Sudoku(int basesize, SymbolType stype, CALLBACK callback): moves_{0}, stype_{stype}, ca
    stats_.basesize = basesize;
    board_len_ = basesize * basesize;
    board_ = new char[stats_.basesize];
}

Sudoku::~Sudoku() {
    delete[] board_;
}

void Sudoku::SetupBoard(const char *values, size_t size) {
    // fill board with values
    for (size_t i=0; i<size; ++i) {
        if (values[i] == '.')
            board_[i] = ' ';
        else
            board_[i] = values[i];

    }
}

bool Sudoku::Solve() {
```

```cpp
    unsigned x = 0;
    unsigned y = 0;

    callback_(*this, board_, MessageType::MSG_STARTING, moves_, stats_.basesize, -1, 0);

    auto success = place_value(x, y);
    if (success)
        callback_(*this, board_, MessageType::MSG_FINISHED_OK, moves_, stats_.basesize, -1, 0)
    else
        callback_(*this, board_, MessageType::MSG_FINISHED_FAIL, moves_, stats_.basesize, -1,

    return success;
}

bool Sudoku::place_value(unsigned x, unsigned y) {
    // base case is when reach 1 after last row
    if (y == board_len_)
        return true;

    // get the 1d index into the board
    unsigned index = x + board_len_ * y;

#ifdef DEBUG_VALID
    cout << "place_value: starting placement of (" << x << "," << y << ") ";
#endif

    // check if pos already occupied
    if (board_[index] != ' ') {

#ifdef DEBUG_VALID
        cout << "NOT PLACED (OCCUPIED), going next pos" << endl;
#endif

        // recurse to next position
        if (x == board_len_ - 1) {
            if (place_value(0, y+1))
                return true;
        }
        else {
            if (place_value(x+1, y))
                return true;
        }
        return false;
    }

    // init the correct type of val
    char val;
    if (stype_ == SymbolType::SYM_NUMBER)
        val = '1';
    else
        val = 'A';

    // loop thru all possible vals and attempt to place them
    for (size_t i=0; i<board_len_; ++i) {
        // check if driver called abort
        if (callback_(*this, board_, MessageType::MSG_ABORT_CHECK, moves_, stats_.basesize, in

#ifdef DEBUG_VALID
            cout << "ABORTED by user" << endl;
#endif

            return false;
        }

        // attempt to place val
        board_[index] = val;
        stats_.moves = ++moves_;
        ++stats_.placed;
        callback_(*this, board_, MessageType::MSG_PLACING, moves_, stats_.basesize, index, val
        if (is_valid(x, y, val)) {

#ifdef DEBUG_VALID
            cout << "VALID POS and PLACING " << val << endl;
#endif

            // recurse to next position
            if (x == board_len_ - 1) {
                if (place_value(0, y+1)) {
                    return true;
                }
            }
            else {
                if (place_value(x+1, y)) {
                    return true;
                }
            }

            // all vals exhausted so backtrack
            board_[index] = ' ';
            //--stats_.placed;
            ++stats_.backtracks;
            callback_(*this, board_, MessageType::MSG_REMOVING, moves_, stats_.basesize, index
        }

        // next val
        board_[index] = ' ';
        --stats_.placed;
        callback_(*this, board_, MessageType::MSG_REMOVING, moves_, stats_.basesize, index, va
        ++val;
    }

    return false;
}

```

```cpp
bool Sudoku::is_valid(unsigned x, unsigned y, char val) {
    unsigned index = x + board_len_ * y;

#ifdef DEBUG_VALID
    cout << "is_valid: validating insert of " << val << " in (" << x << "," << y << ") ";
#endif

    // check if same values in row and col
    for (size_t i=0; i<board_len_; ++i) {

//#ifdef DEBUG_VALID
        //cout << "is_valid: comparing val " << val << " with (" << x << "," << i << ")=" << bo
//#endif
        unsigned next_col_slot_index = i + board_len_ * y;
        unsigned next_row_slot_index = x + board_len_ * i;
        if (    ((next_row_slot_index != index) && (board_[next_row_slot_index] == val))
             || ((next_col_slot_index != index) && (board_[next_col_slot_index] == val))) {

#ifdef DEBUG_VALID
            cout << " INVALID (EXISTS ON ROW/COL)" << endl;
#endif

            return false;
        }
    }

    // check remaining 4 values in the sector
    size_t sectorrow = stats_.basesize * (y/stats_.basesize);
    size_t sectorcol = stats_.basesize * (x/stats_.basesize);
    size_t row1 = (y + stats_.basesize - 1) % stats_.basesize;
    size_t row2 = (y + stats_.basesize + 1) % stats_.basesize;
    size_t col1 = (x + stats_.basesize - 1) % stats_.basesize;
    size_t col2 = (x + stats_.basesize + 1) % stats_.basesize;

#ifdef DEBUG_VALID
    //cout << "  testing board_[" <<   ;
#endif

    if (   (board_[(col1+sectorcol) + board_len_ * (row1+sectorrow)] == val)
        || (board_[(col1+sectorcol) + board_len_ * (row2+sectorrow)] == val)
        || (board_[(col2+sectorcol) + board_len_ * (row1+sectorrow)] == val)
        || (board_[(col2+sectorcol) + board_len_ * (row2+sectorrow)] == val) ) {

#ifdef DEBUG_VALID
        cout << " INVALID (EXISTS IN SECTOR)" << endl;
#endif

        return false;
    }

    return true;
}
```

```cpp

const char* Sudoku::GetBoard() const {
    return board_;
}

Sudoku::SudokuStats Sudoku::GetStats() const {
    return stats_;
}
```