fantasy19 / Derp

Code Issues Pull requests Actions Projects ...

e681c03cbb

Derp / 280 / assignment03-Sudoku / Sudoku.cpp

fantasy19 No commit message

1 contributor

Raw Blame

258 lines (214 sloc) | 7.12 KB

```
/******************************************************************************/
/*!
\file   Sudoku.cpp
\author Ang Cheng Yong
\par    email: a.chengyong\@digipen.edu
\par    DigiPen login: a.chengyong
\par    Course: CS280
\par    Programming Assignment #3
\date   25/10/2016
\brief
This file contains the implementation for BList.
*/
/******************************************************************************/

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

/******************************************************************************/
/*!
\fn Sudoku::Sudoku(int basesize, SymbolType stype , CALLBACK callback )
\brief
constructor for sudoku and sets choices of board values
\return
None
*/
/******************************************************************************/


Sudoku::Sudoku(int basesize, SymbolType stype , CALLBACK callback ) :board(0), cb(callback) {

        // setting statistics
        sStats.basesize = basesize;
        sStats.backtracks =
        sStats.moves =
        sStats.placed = 0;

        // setting values for sudoku puzzle
        first = (stype) ? 'A' : '1';
        width = basesize*basesize;

        last = static_cast<char>(first + static_cast<char>( width-1 ));

}

/****************************************************************************/
/*!
\fn Sudoku::~Sudoku()
\brief
destructor for sudoku, free up the board.
\return
None
*/
/****************************************************************************/

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

/****************************************************************************/
/*!
\fn Sudoku::SudokuStats Sudoku::GetStats() const
\brief
Gettor for sudoku statistics.
\return
data structure for statistics for sudoku board.
*/
/****************************************************************************/

Sudoku::SudokuStats Sudoku::GetStats() const {
        return sStats;
}

/****************************************************************************/
/*!
\fn const char * Sudoku::GetBoard() const
\brief
Gettor for sudoku board
\return
array of values within the board.
*/
/****************************************************************************/

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

```

/******************************************************************************/
/*!
\fn void Sudoku::SetupBoard(const char *values, size_t size)
\brief
Setting up the board with the specified board values and board size
\return
None
*/
/******************************************************************************/

void Sudoku::SetupBoard(const char *values, size_t size) {

        board = new char[size];

        //setting board values
    for (size_t i = 0; i < size; ++i) {
        board[i] = (values[i] == '.') ? EMPTY_CHAR : values[i];
    }
}

/******************************************************************************/
/*!
\fn bool Sudoku::Solve()
\brief
Attempts to solve the sudoku board
\return
True if solvable, false if not
*/
/******************************************************************************/

bool Sudoku::Solve() {

        cb(*this, board, MSG_STARTING, sStats.moves, sStats.basesize, 0, 0);

    size_t init_val = 0;
        if (place_value(init_val)) {
                cb(*this, board, MSG_FINISHED_OK, sStats.moves, sStats.basesize, 0, 0);
                return true;
        }
        else {
                cb(*this, board, MSG_FINISHED_FAIL, sStats.moves, sStats.basesize, 0, 0);
                return false;
        }
}

/******************************************************************************/
/*!
\fn bool Sudoku::place_value(size_t place)
\brief
recursive completes the board by placing values cell after cell or
decides if the board is unsolvable
```

```
\param place
index in the array representing the board to put value in
\return
True if solvable, false if not
*/
/******************************************************************************/

bool Sudoku::place_value(size_t place) {

    if (place == (width*width))
        return true;

        if (board[place] != EMPTY_CHAR)
                return place_value(place + 1);

        for (char val = first; val <= last; ++val) {
                bool abort = cb(*this, board, MSG_ABORT_CHECK, sStats.moves, sStats.basesize, s

                if (abort)
                        return false;

                //placing a value
                board[place] = val;
                ++moves_;
                ++sStats.moves;
                ++sStats.placed;

                cb(*this, board, MSG_PLACING, sStats.moves, sStats.basesize, static_cast<unsign
                // see whether value is valid
                if (ConflictCheck(place, board[place])) {

                        if (place == (width*width) - 1) // stop checking if we're done
                                return true;
                        else {
                                if (place_value(place + 1)) // continue checking if available
                                        return true;

                                if (abort)
                                        return false;
                        }


                }
                else { // replace value if value is invalid and there are still values availabl
                        board[place] = EMPTY_CHAR;
                        --sStats.placed;
                        cb(*this, board, MSG_REMOVING, sStats.moves, sStats.basesize, static_ca
                }

        }
        // all values tried, going back to previous cell to change value to try again
    board[place] = EMPTY_CHAR;
    ++sStats.backtracks;
```

```
        --sStats.placed;
    --moves_;
        return false;
}

/******************************************************************************/
/*!
\fn bool Sudoku::ConflictCheck(size_t place, char val)
\brief
test a value if its valid by checking for conflict in row,
column and the box its in.
\param place
index in the array representing the board to put value in
\param val
value to be tested
\return
true if value is valid, if not false
*/
/******************************************************************************/

bool Sudoku::ConflictCheck(size_t place, char val) {
    bool maintain = true;

    size_t startrow = place - place%width;
    size_t endrow = startrow + width;

    //row check
    for (size_t i = startrow; i < endrow; ++i) {
        if (board[i] == val && i != place) {
            maintain = false;
            break;
        }
    }

    size_t startcol = place % width;
    size_t endcol = startcol + (width - 1)*width;

    //column check
    for (size_t i = startcol; i <= endcol; i += width) {
        if (board[i] == val && i != place) {
            maintain = false;
            break;
        }
    }

        // setting up start and end of value for box check
        // decrementing the row aspect to correct starting position
    size_t boxstart = place - (place % sStats.basesize);

    size_t rowoffset = place / width;

        // decrementing the column aspect to correct starting position
    boxstart -= (rowoffset % sStats.basesize) * width;
```

```cpp

        // end value is the start of last row of box to be checked
    size_t boxend = boxstart + (sStats.basesize - 1) * width;

    // box check
    for (size_t i = boxstart; i <= boxend; i += width) {
        bool brake = true;
        for (size_t j = 0; j <= sStats.basesize - 1; ++j) {
            if (board[i + j] == val && (i + j) != place) {
                brake =
                maintain = false;
                break;
            }
        }
        if (!brake) break;
    }

        // final return value after passing through checks
    return maintain;
}
```