cs380su21-meta.sg

Grade

Reviewed on Tuesday, 6 July 2021, 11:23 PM by Automatic grade
grade: 100.00 / 100.00

Assessment report [-]
[+]**Summary of tests**

Submitted on Tuesday, 6 July 2021, 11:23 PM (Download)

functions.cpp

```
/*!*****************************************************************************
\file functions.cpp
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 8
\date 07-06-2021
\brief
This file has declarations and definitions that are required for submission
*****************************************************************************/

#include "functions.h"

namespace AI
{


} // end namespace
```

functions.h

```cpp
/*!*************************************************************************
\file functions.h
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: B
\par Programming Assignment 8
\date 07-06-2021
\brief
This file has declarations and definitions that are required for submission
*************************************************************************/

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <stack>
#include <algorithm>
#include <cmath>

#include "data.h"

#define UNUSED(x) (void)x;

class NextLocation_Sudoku1D
{
    AI::MapInt1D* map;

public:

    NextLocation_Sudoku1D(void* map)
        : map{ static_cast<AI::MapInt1D*>(map) }{}

    /*!*************************************************************************
    \brief
    A domain specific functor class that is used to find a solution for a
    one-dimensional Sudoku by using Backtracking algorithm.

    \return
    Returns next location which is not occupied on the map. If no location,
    returns Location<> {nullptr, 0};
    *************************************************************************/
    AI::Location<> operator()() const
    {
        for (int i = 0; i < map->size; ++i)
        {
            if (map->base[i] == 0)
                return AI::Location<int>(map->base, i);
        }

        return AI::Location<int>{ nullptr, 0 };
    }
};

class NextLocation_Sudoku2D
{
    AI::MapInt2D* map;

public:

    NextLocation_Sudoku2D(void* map)
        : map{ static_cast<AI::MapInt2D*>(map) }{}

    /*!*************************************************************************
    \brief
    A domain speficifc functor class that is used to find a solution for a
    two-dimensional Sudoku by using Backtracking algorithm.

    \return
    Returns next location which is not occupied on the map. If no location,
    returns Location<> {nullptr, 0};
    *************************************************************************/
    AI::Location<> operator()() const
    {
        for (int j = 0; j < map->height; ++j)
        {
            for (int i = 0; i < map->width; ++i)
            {
                if (map->base[j * map->width + i] == 0)
                    return AI::Location<int>(map->base, j * map->width + i);
            }
        }

        return AI::Location<>{ nullptr, 0 };
    }
};

class NextCandidate_Sudoku1D
{
    AI::MapInt1D* map;

public:

    NextCandidate_Sudoku1D(void* map)
        : map{ static_cast<AI::MapInt1D*>(map) }{}

    /*!*************************************************************************
    \brief
    A domain specific functor class that is used to find a solution for a
    one-dimensional Sudoku by using Backtracking algorithm.

    \param location
    Template class for a location on the map used by Backtracking algorithm.

    \return
    Returns next candidate for a specified location on the map. If no
    candidate found, returns 0.
    *************************************************************************/
```

```
    int operator()(AI::Location<> location)
    {
        int value = map->base[location.getIndex()];

        while (value < map->size)
        {
            value++;
            bool valueFound = false;

            for (int i = 0; i < map->size; ++i)
            {
                if (map->base[i] == value)
                    valueFound = true;
            }

            if (!valueFound)
                return value;
        }

        return 0;
    }
};

class NextCandidate_Sudoku2D
{
    AI::MapInt2D* map;

public:

    NextCandidate_Sudoku2D(void* map)
        : map{ static_cast<AI::MapInt2D*>(map) }{}

    /*!*************************************************************************
    \brief
    A domain specific functor class that is used to find a solution for a
    two-dimensional Sudoku by using Backtracking algorithm.

    \param starting
    Template class for a location on the map used by Backtracking algorithm.

    \return
    Returns next candidate for a specified location on the map. If no
    candidate found, returns 0.
    *************************************************************************/
    int operator()(AI::Location<> location)
    {
        int index = location.getIndex();

        int x = index % map->width;
        int y = index / map->height;

        int value = map->base[y * map->width + x];

        int row_start = y * map->width;

        while (value < (map->width))
        {
            value++;
            bool rowFound = false, columnFound = false, boxFound = false;

            //Check row
            for (int curr_grid = row_start; curr_grid < row_start + map->width;
            ++curr_grid)
            {
                if (curr_grid == index)
                    continue;

                if (map->base[curr_grid] == value)
                    rowFound = true;
            }

            //Check column
            for (int i = 0; i < map->height; ++i)
            {
                int curr_grid = i * map->width + x;

                if (curr_grid == index)
                    continue;

                if (map->base[curr_grid] == value)
                    columnFound = true;
            }

            //Check box
            int box_width = std::sqrt(map->width);
            int box_height = std::sqrt(map->height);

            int startX = x - (x % box_width);
            int startY = y - (y % box_height);

            for (int i = 0; i < box_width; ++i)
            {
                for (int j = 0; j < box_height; ++j)
                {
                    int curr_grid = ((startY + i) * map->width) + startX + j;

                    if (curr_grid == index)
                        continue;

                    if (map->base[curr_grid] == value)
                        boxFound = true;
                }
            }

            if (!rowFound && !columnFound && !boxFound)
                return value;
        }

```

```
        return 0;
    }
};


namespace AI
{
    // Template class of the Backtracking algorithm.
    // Parameter NL defines domain-specific NextLocation functor.
    // Parameter NC defines domain-specific NextCandidate functor.
    template<typename NL, typename NC>
    class Backtracking
    {
        std::stack<Location<>> stack;
        NL next_location;
        NC next_candidate;

    public:
        Backtracking(void* map = nullptr)
            : next_location{ map }, next_candidate{ map }{}

        /*!*********************************************************************
        \brief
        Runs the solve function to find solution in a blocking mode
        *********************************************************************/
        void run()
        {
            while (!solve()) {}
        }

        /*!*********************************************************************
        \brief
        One iteration of the search. Used by run() in a blocking running mode or
        can be called by timer in an non-blocking run

        \return
        Returns true if location is not found. Else, return false if Sudoku is
        not solved.
        *********************************************************************/
        bool solve()
        {
            if (stack.empty())
            {
                AI::Location<> location = next_location();

                if (location.notFound())
                    return true;

                stack.push(location);
            }

            AI::Location<> location = stack.top();
            int candidate = next_candidate(location);

            if (candidate)
            {
                location.setValue(candidate);
                AI::Location<> next = next_location();

                if (next.notFound())
                    return true;

                stack.push(next);
            }
            else
            {
                location.clearValue();
                stack.pop();
            }

            return false;
        }
    };

} // end namespace

#endif
```

VPL

◄ Showcase: Sudoku

Jump to...

Showcase: Tic-Tac-Toe Game ►