cs380su21-meta.sg

Grade

Reviewed on Thursday, 15 July 2021, 4:47 PM by Automatic grade
grade: 100.00 / 100.00

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

Submitted on Thursday, 15 July 2021, 4:47 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 9
\date 07-14-2021
\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 9
\date 07-14-2021
\brief
This file has declarations and definitions that are required for submission
*************************************************************************/

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <vector>
#include <cstring> // memcpy
#include <limits.h>

#include "data.h"

#define UNUSED(x) (void)x;

// Class that defines the game specific data and code
class Grid
{
    static const int width = 3;
    static const int height = 3;

    char squares[width * height];

public:
    // The game marks/pieces
    static const char x = 'x';
    static const char o = 'o';
    static const char _ = ' ';

    Grid(char* squares = nullptr)
        : squares{ _, _, _, _, _, _, _, _, _ }
    {
        if (squares)
            for (int i = 0; i < height * width; ++i)
                this->squares[i] = squares[i];
    }

    Grid(const Grid& rhs)
    {
        this->operator=(rhs);
    }

    void operator=(const Grid& rhs)
    {
        std::memcpy(squares, rhs.squares, height * width * sizeof(char));
    }

    void set(int i, char c)
    {
        squares[i] = c;
    }

    void clear(int i)
    {
        squares[i] = _;
    }

    char* getSquares()
    {
        return squares;
    }

    /*!*************************************************************************
    \brief
    Function loops through a vector array to search through all empty squares
    in the grid and returns a vector of indices.

    For example, grid [' ',' ','o',' ','x',' ',' ',' ',' '], the function will
    return [0,1,3,5,6,7,8]

    \return
    Returns a vector of the empty squares indexs.
    *************************************************************************/
    std::vector<int> emptyIndices() const
    {
        std::vector<int> array = {};

        for (int i = 0; i < width * height; ++i)
        {
            if (this->squares[i] == ' ')
                array.push_back(i);
        }

        return array;
    }

    /*!*************************************************************************
    \brief
    Function to determine player's Tic Tac Toe winning condition.

    \param player
    Player's game mark / pieces

    \return
    Returns true if the grid has a winning configuration for the player, else
    return false.
    *************************************************************************/
    bool winning(char player)
    {
```

```
        return (this->squares[0] == player && this->squares[1] == player &&
                this->squares[2] == player) ||

                (this->squares[3] == player && this->squares[4] == player &&
                 this->squares[5] == player) ||

                (this->squares[6] == player && this->squares[7] == player &&
                 this->squares[8] == player) ||

                (this->squares[0] == player && this->squares[3] == player &&
                 this->squares[6] == player) ||

                (this->squares[1] == player && this->squares[4] == player &&
                 this->squares[7] == player) ||

                (this->squares[2] == player && this->squares[5] == player &&
                 this->squares[8] == player) ||

                (this->squares[0] == player && this->squares[4] == player &&
                 this->squares[8] == player) ||

                (this->squares[2] == player && this->squares[4] == player &&
                 this->squares[6] == player);
    }

    /*!*************************************************************************
    \brief
     An overloading insertion operator function that takes and return a stream
     object.

    \param os
    Output stream to perform output.

    \param rhs
    Right hand side object.

    \return
    Returns the output through ostream.
    *************************************************************************/
    friend std::ostream& operator<<(std::ostream& os, const Grid& rhs)
    {
        for (int j = 0; j < height; ++j)
        {
            if (j == 0)
                os << "[";
            else
                os << " ";

            for (int i = 0; i < width; ++i)
            {
                os << rhs.squares[j * width + i];

                if (i == width - 1 && j == height - 1)
                    os << "]";
                else
                    os << ",";
            }

            if (j != height - 1)
                os << std::endl;
        }

        return os;
    }
};

namespace AI
{
    // A node of the game tree
    template<typename T>
    class Move
    {
        T grid;         // Result of a move: new state of the game grid
        int score;      // Score of the move
        std::vector<Move*>* next;  // All possible next moves

        // Index of the first move in member next that has the best score
        int bestMove;
        int spotIndex;  // Index of the move's spot (used for a visualization)

    public:
        Move(T grid = {}, int score = 0,
            std::vector<Move*>* next = new std::vector<Move*>{},
            int bestMove = -1)
            : grid{ grid }, score{ score }, next{ next }, bestMove{ bestMove },
            spotIndex{ -1 }{}

        /*!*************************************************************************
        \brief
        Destructor for class Move
        *************************************************************************/
        ~Move()
        {
            //Delete each move object in vector next
            for (size_t i = 0; i < next->size(); ++i)
            {
                delete (*next)[i];
            }

            //Remove vector elements, make vector size to 0.
            next->clear();

            //Delete pointer to vector
            delete next;
        }

        /*!*************************************************************************
```

```
        \brief
        Function to move object to specific index location

        \param i
        Index position to be moved to

        \return
        Returns a reference to Move object
        ********************************************************************/
        Move& at(int i)
        {
            return *((*next)[i]);
        }

        int getScore() const
        {
            return score;
        }

        void setSpotIndex(int i)
        {
            spotIndex = i;
        }

        friend std::ostream& operator<<(std::ostream& os, const Move& rhs)
        {
            os << rhs.grid << std::endl;
            os << rhs.score << std::endl;
            os << rhs.next->size() << std::endl;
            os << rhs.bestMove << std::endl;
            return os;
        }
    };

     /*!*******************************************************************
     \brief
     Function to find the best move for maximizer. For initial call, set player
     parameter as maximizer. However, found solution (sequence of moves) may not
     be necessary the shortest.

     \param grid
     Current state of the Tic Tac Toe board.

     \param player
     Player's game mark / pieces

     \param maximizer
     Maximizer's game mark / pieces

     \param minimizer
     Minimizer's game mark / pieces

     \return
     Returns a pointer to Move object
     ********************************************************************/
    template<typename T>
    Move<T>* minimax(T grid, char player, char maximizer, char minimizer)
    {
        if (grid.winning(minimizer))
            return new Move<T>(grid, -10 );
        else if (grid.winning(maximizer))
            return new Move<T>(grid, 10);

        std::vector<int> availSpots = grid.emptyIndices();

        if (availSpots.size()== 0)
            return new Move<T>(grid, 0);

        std::vector<Move<T>*>* next_ = new std::vector<Move<T>*>;

        for (size_t i = 0; i < availSpots.size(); ++i)
        {
            grid.set(availSpots[i], player);

            AI::Move<T>* move = minimax(grid,
                (player == maximizer) ? minimizer : maximizer,
                maximizer,
                minimizer);

            move->setSpotIndex(availSpots[i]);
            next_->push_back(move);
            grid.clear(availSpots[i]);
        }

        int bestScore_ = 0;
        int bestMove_ = -1;

        if (player == maximizer)
        {
            bestScore_ = -INT_MAX;

            for (size_t i = 0; i < next_->size(); ++i)
            {
                if ((*next_)[i]->getScore() > bestScore_)
                {
                    bestScore_ = (*next_)[i]->getScore();
                    bestMove_ = i;
                }
            }
        }
        else
        {
            bestScore_ = INT_MAX;

            for (size_t i = 0; i < next_->size(); ++i)
            {
                if ((*next_)[i]->getScore() < bestScore_)
                {
```

```
                    bestScore_ = (*next_)[i]->getScore();
                    bestMove_ = i;
                }
            }
        }

        // Return the move
        return new Move<T>(grid, bestScore_, next_, bestMove_);
    }

} // end namespace

#endif
```

VPL

◄ Showcase: Tic-Tac-Toe Game

Jump to...

Minimax. Example of a quiz question ►