```
/*
 * File:   minimax.h
 * Author: Taylor Cowley
 *
 * Created on May 28, 2015, 12:40 PM
 */

#include "minimax.h"
#include <stdio.h>
#include <stdint.h>

//This searches through our board of scores to find where the highest one is
minimax_move_t minimax_getHighestScoreMove(minimax_board_t* scores);

//This searches through our board of scores to find where the least one is.
minimax_move_t minimax_getLowestScoreMove(minimax_board_t* scores);


//This holds all the temporary best moves until we land on the ultimate best move.
minimax_move_t choice;


//This is our recursive function that computes what the best move is!
int8_t minimax(minimax_board_t* board, bool player, int8_t depth) {

    //So we calculate the final board score
    minimax_score_t score = minimax_computeBoardScore(board, !player);

    //if the game is over
    if (minimax_isGameOver(score)) {
        return player ? score + depth : score - depth ; //we return the final score.
    }

    //We need to go deeper
    depth = depth + 1;

    //This stores the possible scores for every possible move.
    minimax_board_t move_scores;

    // Otherwise, you need to recurse.
    // This while-loop will generate all possible boards.
    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
            //we can only make moves on empty squares
            if (board->squares[row][col] == MINIMAX_EMPTY_SQUARE) {
                if (player) {   //The player gets a player square
                    board->squares[row][col] = MINIMAX_PLAYER_SQUARE;
                } else {        //the opponent gets an opponent square
                    board->squares[row][col] = MINIMAX_OPPONENT_SQUARE;
                }
                //RECURSIVELY compute the score for this move
                score = minimax(board, !player, depth);
                //record that score.
                move_scores.squares[row][col] = score;
                // Undo the change to the board  prior to next iteration of for-loops.
                board->squares[row][col] = MINIMAX_EMPTY_SQUARE;
            } else {
                //Someone is already there! it is an invalid move.
                move_scores.squares[row][col] = MINIMAX_INVALID_MOVE;
            }
        }
    }

    // Once you get here, you have iterated over empty squares at this level. All of the scores
computed
    // in the move-score table for boards at this level.
    // Now you need to return the score depending upon whether you are computing min or max.
    if (player) {
        //If we are player we are looking for the highest score
        choice = minimax_getHighestScoreMove(&move_scores);
        //record the score
        score = move_scores.squares[choice.row][choice.column];
    } else {
        //We are the opponent! We are looking for the lowest score
        choice = minimax_getLowestScoreMove(&move_scores);
        //record the score
        score = move_scores.squares[choice.row][choice.column];
    }
    //Woo! Return the best score.
    return score;
}

//This searches through our board of scores to find where the highest one is
minimax_move_t minimax_getHighestScoreMove(minimax_board_t* scores) {
    //The currently highest score isn't.
    int8_t highestScore = MINIMAX_INVALID_MOVE;
    //we also record the highest move
    minimax_move_t highestMove;

    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
            //So if we haven't recorded a score OR if this score is higher than our previously
recorded higher score
            if (highestScore == MINIMAX_INVALID_MOVE || (scores->squares[row][col] !=
MINIMAX_INVALID_MOVE && scores->squares[row][col] > highestScore)) {
                highestMove.column = col;                   //record the column of highest move
                highestMove.row = row;                      //record the row of highest score
                highestScore = scores->squares[row][col];   //and record the currently highest
score
            }
        }
    }
    //And we return the best move.
    return highestMove;
}

//This searches through our board of scores to find where the least one is.
minimax_move_t minimax_getLowestScoreMove(minimax_board_t* scores) {
    //Initializing the variables
    int8_t lowestScore = MINIMAX_INVALID_MOVE;
    minimax_move_t lowestMove;

    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
```

```
            //So if we haven't recorded a score yet OR this score is higher than our previously
recorded higher score
            if (lowestScore == MINIMAX_INVALID_MOVE || (scores->squares[row][col] !=
MINIMAX_INVALID_MOVE && scores->squares[row][col] < lowestScore)) {
                lowestMove.column = col;                    //record the column
                lowestMove.row = row;                       //record the row
                lowestScore = scores->squares[row][col];    //record the currently lowest score
            }
        }
    }
    //Woo! return it!
    return lowestMove;
}

// Determine that the game is over by looking at the score.
bool minimax_isGameOver(minimax_score_t score) {

    //if there is a score at all, then the game is not over.
    if (score == MINIMAX_NOT_ENDGAME) {
        return false;
    }

    //Hey! there is a score number. it is endgame.
    return true;
}


// Returns the score of the board, based upon the player.
int16_t minimax_computeBoardScore(minimax_board_t* board, bool player) {

    //printf("this is the board we are receiving in computeBoardScore\n\r");
    //print_board(board, true);
    //check to see if all 3 columns are the same in any row.
    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        //check if all of the columns are the same and team "empty" hasn't won
        if (board->squares[row][0] != MINIMAX_EMPTY_SQUARE
                && board->squares[row][0] == board->squares[row][1]
                && board->squares[row][0] == board->squares[row][2]
                && board->squares[row][1] == board->squares[row][2]) {
            //Woo! Someone won!
            //if it was the player that played those and we are calculating it for the player,
            //the player wins. Else the opponent wins
            return (board->squares[row][0] == MINIMAX_PLAYER_SQUARE) && player
                    ? MINIMAX_PLAYER_WINNING_SCORE : MINIMAX_OPPONENT_WINNING_SCORE;
        }
    }

    //check to see if all 3 rows are the same in any columns.
    for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
        //check if all of the rows are the same and team "empty" hasnt won
        if (board->squares[0][col] != MINIMAX_EMPTY_SQUARE
                && board->squares[0][col] == board->squares[1][col]
                && board->squares[0][col] == board->squares[2][col]
                && board->squares[1][col] == board->squares[2][col]) {
            //Woo! Someone won!
            //if it was the player that played those and we are calculating it for the player,
            //the player wins. Else the opponent wins
```

```c
            return (board->squares[0][col] == MINIMAX_PLAYER_SQUARE) && player
                ? MINIMAX_PLAYER_WINNING_SCORE : MINIMAX_OPPONENT_WINNING_SCORE;
        }
    }

    //checks the top left to bottom right diagonal
    if (board->squares[0][0] != MINIMAX_EMPTY_SQUARE
            && board->squares[0][0] == board->squares[1][1]
            && board->squares[0][0] == board->squares[2][2]
            && board->squares[1][1] == board->squares[2][2]) {
        //Woo! Someone won!
        //if it was the player that played those and we are calculating it for the player,
        //the player wins. Else the opponent wins
        return (board->squares[0][0] == MINIMAX_PLAYER_SQUARE) && player
            ? MINIMAX_PLAYER_WINNING_SCORE : MINIMAX_OPPONENT_WINNING_SCORE;
    }

    //checks the bottom left to top right diagonal
    if (board->squares[0][2] != MINIMAX_EMPTY_SQUARE
            && board->squares[0][2] == board->squares[1][1]
            && board->squares[0][2] == board->squares[2][0]
            && board->squares[1][1] == board->squares[2][0]) {
        //Woo! Someone won!
        //if it was the player that played those and we are calculating it for the player,
        //the player wins. Else the opponent wins
        return (board->squares[0][2] == MINIMAX_PLAYER_SQUARE) && player
            ? MINIMAX_PLAYER_WINNING_SCORE : MINIMAX_OPPONENT_WINNING_SCORE;
    }

    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
            //check to make sure all of the squares are not filled
            if (board->squares[row][col] == MINIMAX_EMPTY_SQUARE) //we have an empty square
                return MINIMAX_NOT_ENDGAME; //that means we have not finished the game
        }
    }
    //All squares are filled but no one has one. Catsgame!
    return MINIMAX_DRAW_SCORE;
}

// Init the board to all empty squares.
void minimax_initBoard(minimax_board_t* board) {
    for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
        for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
            board->squares[row][col] = MINIMAX_EMPTY_SQUARE;
        }
    }
}


// This routine is not recursive but will invoke the recursive minimax function.
// It computes the row and column of the next move based upon:
// the current board,
// the player. true means the computer is X. false means the computer is O.
void minimax_computeNextMove(minimax_board_t* board, bool player, uint8_t* row, uint8_t* column)
{
    //So let us compute it! minimax puts its move into choice.
    minimax(board, player, 0);

    *column = choice.column; //retrieve move from choice
    *row = choice.row; //retrieve move from choice
}

//This printfs the board. if sanitize, it will say X and O instead of numbers
void minimax_print_board(minimax_board_t* board, bool sanitize) {
    for (int8_t col = 0; col < MINIMAX_BOARD_COLUMNS; col++) { //cycle through the column
        for (int8_t row = 0; row < MINIMAX_BOARD_ROWS; row++) { //cycle through the row
            if (!sanitize) {    //We want numbers!
                printf("%d ", board->squares[row][col]);
            } else {
                //We want it to say X or O
                if (board->squares[row][col] == MINIMAX_PLAYER_SQUARE) {//player is X
                    printf("X");
                } else if (board->squares[row][col] == MINIMAX_OPPONENT_SQUARE) {//opponent is O
                    printf("O");
                } else {    //Having problems with spaces, so empty squares are .
                    printf(".");
                }
            }
        }
        //A nice space at the end
        printf("\n\r");
    }
}
```