```c
//  minimax.c

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

#define ROW_0 0 //top row
#define ROW_1 1 //middle row
#define ROW_2 2 //bottom row

#define COLUMN_0 0     //left column
#define COLUMN_1 1     //middle column
#define COLUMN_2 2     //right column
#define MAX_MOVES 9    //max size of our move-score table
#define FIRST_ENTRY 0  //first row of move-score table
#define SECOND_ENTRY 1 //second row of move-score table

minimax_move_t choice; //final move choice of type minimax_move_t

//recursive algorithm to determine what the best move is
minimax_score_t minimax(minimax_board_t* board, bool current_player_is_x){
    minimax_move_t moves[MAX_MOVES];  //moves char array for move-score table
    minimax_score_t scores[MAX_MOVES];//scores char array for move-score table
    uint8_t tableIndex = FIRST_ENTRY;  //variable used to add entries to move-score table
and then iterate through table
    bool choiceMade = false; //boolean used to determine if we should use the first entry
in the move-score table as the best move
    minimax_score_t score;

    if(minimax_isGameOver(minimax_computeBoardScore(board, current_player_is_x))){ //base
case of recursive function is to see if the game is over
        return minimax_computeBoardScore(board, current_player_is_x); //return current
score if game IS over
    }

    for(int r = ROW_0; r < MINIMAX_BOARD_ROWS; r++){ //iterate through rows
        for(int c = COLUMN_0; c < MINIMAX_BOARD_COLUMNS; c++){ //iterate through columns
            if(board->squares[r][c] == MINIMAX_EMPTY_SQUARE){ //check if that square is
empty
                if(current_player_is_x) board->squares[r][c] = MINIMAX_X_SQUARE; //if the
player whose turn it is is X's, put an X in the empty square
                else board->squares[r][c] = MINIMAX_O_SQUARE; //if they're O's, put an O
in the empty square

                score = minimax(board, !current_player_is_x); //to see the end results of
that move, we descend another level of recursion, switching the current player
                scores[tableIndex] = score; //put that score returned from minimax into
the move-score table
                moves[tableIndex].row = r;  //the current row and column constitute the
move that should be added to the table
                moves[tableIndex].column = c;
                tableIndex++; //increment table index

                board->squares[r][c] = MINIMAX_EMPTY_SQUARE; //blank out the X or O we
just put since it's all to figure out the best move
            }
        }
    }
```

```
    if(current_player_is_x){
        int i = FIRST_ENTRY;
        score = scores[i]; //initially set scores to the first entry in the table so we
have something to compare to
        for(i = SECOND_ENTRY; i < tableIndex; i++){ //iterate through table starting from
the second entry
            if(score < scores[i]){ //since the current player is X, check if the current
score is less than the next score in the table
                score = scores[i]; //if it is, we'll want that score, so set score equal
to it
                choice = moves[i]; //set the choice equal to the move corresponding to
that score
                choiceMade = true; //we made our move choice, so change this to true
            }
        }
        if(!choiceMade) choice = moves[FIRST_ENTRY]; //if a moves choice was not made,
then we set choice to the first move in the table
    }
    else{ //player is O
        int i = FIRST_ENTRY;
        score = scores[i]; //same initialization as above
        for(i = SECOND_ENTRY; i < tableIndex; i++){ //iterate through starting from
second entry
            if(score > scores[i]){ //since player is O, check if current score is greater
than the next score in the table
                score = scores[i]; //if it is, get that score to return
                choice = moves[i]; //set choice equal to corresponding move
                choiceMade = true; //our move choice has been made
            }
        }
        if(!choiceMade) choice = moves[FIRST_ENTRY]; //if we never set choice to
anything, set it to the first entry in the table
    }

    return score; //we will return the best score depending on whether the current player
is O's or X's
}

//function that calls minimax to figure out the best move
void minimax_computeNextMove(minimax_board_t* board, bool current_player_is_x, uint8_t*
row, uint8_t* column){
    minimax(board, current_player_is_x);

    *row = choice.row;      //after minimax finished, choice will be set to the best
possible move. Get that row
    *column = choice.column;//get the column of choice
}

// Determine that the game is over by looking at the score.
bool minimax_isGameOver(minimax_score_t score){
    if(score == MINIMAX_NOT_ENDGAME) return false; //there's only one thing the score can
equal for it to not be over
    else return true; //if it isn't that value, the game isn't over
}

//helper function to check if there's a win including the middle spot
bool checkForCenterWin(minimax_board_t* board, uint16_t playerNumber){
    if((board->squares[ROW_0][COLUMN_0] == playerNumber && board->squares[ROW_2]
```

```
[COLUMN_2] == playerNumber) || //check for diagonal top to bottom win
       (board->squares[ROW_1][COLUMN_0] == playerNumber && board->squares[ROW_1]
[COLUMN_2] == playerNumber) || //check for left to right horizontal win
       (board->squares[ROW_2][COLUMN_0] == playerNumber && board->squares[ROW_0]
[COLUMN_2] == playerNumber) || //check for diagonal bottom to top win
       (board->squares[ROW_0][COLUMN_1] == playerNumber && board->squares[ROW_2]
[COLUMN_1] == playerNumber)) //check for vertical win
        return true; //if any of those wins happened, we return true
    return false;
}

//helper function to check if there's a win not using the middle spot on the board
bool checkForBorderWin(minimax_board_t* board, uint8_t playerNumber){

    if(board->squares[ROW_0][COLUMN_0] == playerNumber){ //half the border win
possibilities require top left corner
        if(((board->squares[ROW_1][COLUMN_0] == playerNumber) && (board->squares[ROW_2]
[COLUMN_0] == playerNumber)) || //checks for top to bottom win on left column
           ((board->squares[ROW_0][COLUMN_1] == playerNumber) && (board->squares[ROW_0]
[COLUMN_2] == playerNumber)))   //checks for horizontal win on top row
            return true;
    }
    if(board->squares[ROW_2][COLUMN_2] == playerNumber){ //the other half of border wins
require the bottom right corner
        if(((board->squares[ROW_2][COLUMN_0] == playerNumber) && (board->squares[ROW_2]
[COLUMN_1] == playerNumber)) || //checks for horizontal win on bottom row
           ((board->squares[ROW_0][COLUMN_2] == playerNumber) && (board->squares[ROW_1]
[COLUMN_2] == playerNumber)))   //checks for vertical win on right column
            return true;
    }
    return false; //if none of those combinations occurred, there was definitely not a
border win
}

//helper function to see if the board is full. Used to check for draws
bool checkIfBoardFull(minimax_board_t* board){
    for(int i = 0; i < MINIMAX_BOARD_ROWS; i++) //iterate through the rows
        for(int j = 0; j < MINIMAX_BOARD_COLUMNS; j++) //iterate through columns
            if(board->squares[i][j] == MINIMAX_EMPTY_SQUARE) //if any square is empty, we
immediately know the board is not full and can return
                return false;

    return true; //if we get through the whole board without having returned, the board
must be full
}

//needed to determine based on what spots are filled with what letter the score of the
current board
minimax_score_t minimax_computeBoardScore(minimax_board_t* board, bool player_is_x){

    if(((board->squares[ROW_1][COLUMN_1] == MINIMAX_X_SQUARE) && checkForCenterWin(board,
MINIMAX_X_SQUARE)) || //check if the center spot has an X, then check for a center win
using the function
            checkForBorderWin(board, MINIMAX_X_SQUARE)) //if there's no center win, check
for a border win
        return MINIMAX_X_WINNING_SCORE; //if either of those win scenarios is true, then
X won
    else if((board->squares[ROW_1][COLUMN_1] == MINIMAX_O_SQUARE &&
```

```
checkForCenterWin(board, MINIMAX_O_SQUARE)) || //check for O in center spot, then for
center win using function
            checkForBorderWin(board, MINIMAX_O_SQUARE)) //if not, check for a border O win
        return MINIMAX_O_WINNING_SCORE; //if either of the scenarios was true, then O won
    else if(checkIfBoardFull(board)) return MINIMAX_DRAW_SCORE; //if those wins aren't
true, we check if the board is full, which would mean a draw
    else return MINIMAX_NOT_ENDGAME; //if not a draw or an O win, then the game isn't over

}


// Init the board to all empty squares.
void minimax_initBoard(minimax_board_t* board){
    for(int i = 0; i < MINIMAX_BOARD_ROWS; i++){ //iterate through rows
        for(int j = 0; j < MINIMAX_BOARD_COLUMNS; j++){ //iterate through columns
            board->squares[i][j] = MINIMAX_EMPTY_SQUARE; //make each square empty
        }
    }
}
```