```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
        }
    }
}
```

```
 * ticTacToeDisplay.c

#include "ticTacToeDisplay.h"
#include "supportFiles/utils.h"                 //needed to use the delays function
#include "../Lab2_SwitchesAndButtons/buttons.h" //needed to use buttons 0 and 1
#include "../Lab2_SwitchesAndButtons/switches.h"//needed to use switch 0
#include "stdio.h"

#define LEFT_VERT_LINE DISPLAY_WIDTH/3          //x coordinate for left vertical line
#define RIGHT_VERT_LINE 2*DISPLAY_WIDTH/3       //x coordinate for right vertical line
#define UPPER_HORIZ_LINE DISPLAY_HEIGHT/3       //y coordinate for upper horizontal line
#define LOWER_HORIZ_LINE 2*DISPLAY_HEIGHT/3     //y coordinate for lower horizontal line

#define X1_LEFT_COL DISPLAY_WIDTH/12                    //x for left side of X's in left
column
#define X2_LEFT_COL LEFT_VERT_LINE-DISPLAY_WIDTH/12   //x for right side of X's in left
column
#define X1_MID_COL LEFT_VERT_LINE+DISPLAY_WIDTH/12    //x for left side of X's in middle
column
#define X2_MID_COL RIGHT_VERT_LINE-DISPLAY_WIDTH/12   //x for right side of X's in middle
column
#define X1_RIGHT_COL RIGHT_VERT_LINE+DISPLAY_WIDTH/12 //x for left side of X's in right
column
#define X2_RIGHT_COL DISPLAY_WIDTH-DISPLAY_WIDTH/12   //x for right side of X's in right
column

#define Y1_TOP_ROW DISPLAY_HEIGHT/15                    //y for top of X's in top row
#define Y2_TOP_ROW UPPER_HORIZ_LINE-DISPLAY_HEIGHT/15 //y for bottom of X's in top row
#define Y1_MID_ROW UPPER_HORIZ_LINE+DISPLAY_HEIGHT/15 //y for top of X's in middle row
#define Y2_MID_ROW LOWER_HORIZ_LINE-DISPLAY_HEIGHT/15 //y for bottom of X's in middle row
#define Y1_BOT_ROW LOWER_HORIZ_LINE+DISPLAY_HEIGHT/15 //y for top of X's in bottom row
#define Y2_BOT_ROW DISPLAY_HEIGHT-DISPLAY_HEIGHT/15   //y for bottom of X's in bottom row

#define X_CIR_LEFT_COL DISPLAY_WIDTH/6     //x for midpoint of O's in left column
#define X_CIR_MID_COL DISPLAY_WIDTH/2      //x for midpoint of O's in middle column
#define X_CIR_RIGHT_COL 5*DISPLAY_WIDTH/6  //x for midpoint of O's in right column
#define Y_CIR_TOP_ROW DISPLAY_HEIGHT/6     //y for midpoint of O's in top row
#define Y_CIR_MID_ROW DISPLAY_HEIGHT/2     //y for midpoint of O's in middle row
#define Y_CIR_BOT_ROW 5*DISPLAY_HEIGHT/6   //y for midpoint of O's in bottom row
#define CIR_RADIUS DISPLAY_HEIGHT/10       //radius of all the O's

#define Y_0 0 //y coordinate value of zero
#define X_0 0 //x coordinate value of zero

#define RUNNING 1      //used to keep while loop going
#define TOUCH_DELAY 50 //50 ms, necessary delay after registering a touch

#define COLUMN_0 0 //left column on board
#define COLUMN_1 1 //middle column on board
#define COLUMN_2 2 //right column on board

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

#define TEST_OVER "Yo yo, you done \nfinished dis test\n\r" //message to print when BTN1
is pressed
#define TEXT_SIZE 3 //text size of final message
```

```
// Inits the tic-tac-toe display, draws the lines that form the board.
void ticTacToeDisplay_init(){
    display_init();  // Must init all of the software and underlying hardware for LCD.
    display_fillScreen(DISPLAY_BLACK); // Blank the screen.
    ticTacToeDisplay_drawBoardLines(); //we have to draw the board when starting the game
}

// Draws an X at the specified row and column.
// erase == true means to erase the X by redrawing it as background. erase == false, draw
the X as foreground.
void ticTacToeDisplay_drawX(uint8_t row, uint8_t column, bool erase){
    uint16_t x1, x2, y1, y2; //X's need 4 coordinate values
    if(column == COLUMN_0){ //if touch was in left column, we know what the x-coordinates
need to be
        x1 = X1_LEFT_COL; //these 2 constants are described above
        x2 = X2_LEFT_COL;
    }
    else if(column == COLUMN_1){ //if touch was in middle column, set the corresponding x
coordinates
        x1 = X1_MID_COL; //these 2 constants are described above
        x2 = X2_MID_COL;
    }
    else{ //if it gets here, we know the touch was in the right column
        x1 = X1_RIGHT_COL; //these 2 constants are described above
        x2 = X2_RIGHT_COL;
    }

    if(row == ROW_0){ //check if touch was in top row
        y1 = Y1_TOP_ROW; //these 2 constants are described above
        y2 = Y2_TOP_ROW;
    }
    else if(row == ROW_1){ //check if touch was in middle row
        y1 = Y1_MID_ROW; //these 2 constants are described above
        y2 = Y2_MID_ROW;
    }
    else{ //if it gets here, touch was in bottom row
        y1 = Y1_BOT_ROW; //these 2 constants are described above
        y2 = Y2_BOT_ROW;
    }

    //draw the 2 lines needed to form an x
    if(!erase){
        display_drawLine(x1, y1, x2, y2, DISPLAY_YELLOW);
        display_drawLine(x1, y2, x2, y1, DISPLAY_YELLOW);
    }
    else{
        display_drawLine(x1, y1, x2, y2, DISPLAY_BLACK);
        display_drawLine(x1, y2, x2, y1, DISPLAY_BLACK);
    }
}

// Draws an O at the specified row and column.
// erase == true means to erase the X by redrawing it as background. erase == false, draw
the X as foreground.
void ticTacToeDisplay_drawO(uint8_t row, uint8_t column, bool erase){
    uint16_t x,y; //to draw a circle, we just need x-y coordinate pair for the center of
```

```
the circle and the radius

    if(column == COLUMN_0) x = X_CIR_LEFT_COL; //if touch was in left column, set it to
constant described above
    else if(column == COLUMN_1) x = X_CIR_MID_COL; //if touch was in middle column, set
it to constant described above
    else x = X_CIR_RIGHT_COL; //if touch was in right column, set it to constant
described above

    if(row == ROW_0) y = Y_CIR_TOP_ROW; //top row touch, set it to constant described
above
    else if(row == ROW_1) y = Y_CIR_MID_ROW; //middle row touch, set it to constant
described above
    else y = Y_CIR_BOT_ROW; //bottom row touch, set it to constant described above

    if(!erase) display_drawCircle(x, y, CIR_RADIUS, DISPLAY_YELLOW); //draw the O. Color
yellow
    else display_drawCircle(x, y, CIR_RADIUS, DISPLAY_BLACK);
}

// After a touch has been detected and after the proper delay, this sets the row and
column arguments
// according to where the user touched the board.
void ticTacToeDisplay_touchScreenComputeBoardRowColumn(uint8_t* row, uint8_t* column){
    int16_t x, y; //x-y coordinates of touch
    uint8_t z;    //pressure of touch
    display_getTouchedPoint(&x, &y, &z); //used to get the x,y coordinates for the touch

    //sets row to the horizontal third of the board the touch was in
    if(y <= UPPER_HORIZ_LINE) //check if touch was in top row
        *row = ROW_0;
    else if(y >= LOWER_HORIZ_LINE) //check if touch was in bottom row
        *row = ROW_2;
    else //check if touch was in middle row
        *row = ROW_1;

    //sets column to vertical third of the board the touch was in
    if(x <= LEFT_VERT_LINE) //check if touch was in left column
        *column = COLUMN_0;
    else if(x >= RIGHT_VERT_LINE) //check if touch was in right column
        *column = COLUMN_2;
    else //check if touch was in middle column
        *column = COLUMN_1;
}

// Runs a test of the display. Does the following.
// Draws the board. Each time you touch one of the screen areas, the screen will paint
// an X or an O, depending on whether switch 0 (SW0) is slid up (O) or down (X).
// When BTN0 is pushed, the screen is cleared. The test terminates when BTN1 is pushed.
void ticTacToeDisplay_runTest(){
    switches_init(); //must initialize switches
    buttons_init();  //must initialize buttons
    ticTacToeDisplay_init(); //call to initialize game board
    while(RUNNING){
        uint32_t buttons = buttons_read(); //used to get which of the 4 buttons are being
touched

        if((buttons & BUTTONS_BTN0_MASK) == BUTTONS_BTN0_MASK) { //we want to reset the
```

```
game if BTN0 was pressed
            display_fillScreen(DISPLAY_BLACK); // Blank the screen.
            ticTacToeDisplay_drawBoardLines(); //redraw the game board
        }
        else if((buttons & BUTTONS_BTN1_MASK) == BUTTONS_BTN1_MASK) break; //if BTN1 is
pressed we reset
        if(display_isTouched()){ //check for touch

            uint8_t row, column;
            utils_msDelay(TOUCH_DELAY);  //needed delay after touch is detected
            ticTacToeDisplay_touchScreenComputeBoardRowColumn(&row, &column); //call to
figure out row and column to put O/X in based on touch coordinates
            if(switches_read() & SWITCHES_SW0_MASK)ticTacToeDisplay_drawO(row, column,
false); //if SW0 is in the up position, we draw an O
            else ticTacToeDisplay_drawX(row, column, false); //if the switch is down,
draw an X
        }
    }
    display_fillScreen(DISPLAY_BLACK);  // Blank the screen.
    display_setTextSize(TEXT_SIZE);     //set text size for final message
    display_setTextColor(DISPLAY_GREEN);//final message will be green
    display_println(TEST_OVER);         //print final message
}

// This will draw the four board lines.
void ticTacToeDisplay_drawBoardLines(){
    display_drawLine(LEFT_VERT_LINE, Y_0, LEFT_VERT_LINE, DISPLAY_HEIGHT,
DISPLAY_YELLOW);    //draw left vertical line
    display_drawLine(RIGHT_VERT_LINE, Y_0, RIGHT_VERT_LINE, DISPLAY_HEIGHT,
DISPLAY_YELLOW);  //draw right vertical line
    display_drawLine(X_0, UPPER_HORIZ_LINE, DISPLAY_WIDTH, UPPER_HORIZ_LINE,
DISPLAY_YELLOW); //draw upper horizontal line
    display_drawLine(X_0, LOWER_HORIZ_LINE, DISPLAY_WIDTH, LOWER_HORIZ_LINE,
DISPLAY_YELLOW); //draw lower horizontal line
}
```

```c
 * ticTacToeControl.c

#include "ticTacToeControl.h"
#include "ticTacToeDisplay.h"
#include "minimax.h"
#include "supportFiles/utils.h"                 //needed to use the delays function
#include "../Lab2_SwitchesAndButtons/buttons.h" //needed to use buttons 0 and 1

// States for the controller state machine.
enum ticTacToeControl_st_t {
    init_st,                // Start here, transition out of this state on the first
tick.
    new_game_st,            //come here each time a new game is started
    player_turn_st,         //where the human takes its turn
    comp_turn_st,           //where the computer takes its turn
    waiting_for_touch_st,   //waiting for a touch after the computer's turn
    game_over_st,           //this means the game is over and we wait for a reset
    debounce_st             //state needed to clear old touch data
} currentState;

#define COUNTER_MIN 0   //what counter should be reset to
#define NEW_GAME_MAX 40 //wait time in new game st

#define MAX_ROWS 3    //rows in a board
#define MAX_COLUMNS 3 //columns in a board
#define ROW_0 0       //first row
#define COL_0 0       //first column

#define CURSOR_X 60                 //text cursor x-coordinate
#define CURSOR_Y 2*DISPLAY_HEIGHT/5 //text cursor y-coordinate

#define START_MESSAGE "Touch the board to play X\n\t\t\t\t -or-\n \t\tWait for the
computer and play O"
#define TEXT_SIZE 2
#define START_DELAY 3000 //how long to show start screen

static minimax_board_t board;   //we need a global board that can only be accessed by
this file
static bool newGame;            //boolean to identify if it's a new game or not
static bool playerIsX;          //bool to determine what letter the current player is
static bool wrongTouch;         //bool used to identify touch on already occupied space
on board
static uint16_t newGameCounter; //counter to keep track of how long we want to be in new
game st
static uint8_t row, column;     //we need the row and column for a move.

//init function for state machine
void ticTacToeControl_init(){
    currentState = init_st;  //always set the state to init_st at the beginning
    newGame = true;          //it is a new game
    playerIsX = true;        //X always goes first in tic tac toe
    wrongTouch = false;      //we assume there's no touch on an already occupied space at
first
    newGameCounter = COUNTER_MIN; //initalize counter

    display_init(); //gotta initialize the display
    display_fillScreen(DISPLAY_BLACK); // Blank the screen.
    display_setCursor(CURSOR_X, CURSOR_Y); //set cursor for start screen
```

```
    display_setTextSize(TEXT_SIZE); //set text size
    display_println(START_MESSAGE); //display the start message
    utils_msDelay(START_DELAY); //start screen should be displayed for a certain amount
of time
    ticTacToeDisplay_init(); //draws the tic tac toe board
    buttons_init(); //we're using a button, so we gotta initialize all of them
    minimax_initBoard(&board); //gotta initialize the minimax board
}

//tick function
void ticTacToeControl_tick(){

    //perform transitions first
    switch(currentState){
    case init_st:
        currentState = new_game_st; //go immediately from the init_st to the new_game_st
        break;
    case new_game_st:
        if(display_isTouched()){ //detect touch
            display_clearOldTouchData(); //clear any former touch data
            currentState = debounce_st; //go to debounce state to get correct touch data
            newGame = false; //we can set this to false since the human is taking the
first turn
        }
        else if(newGameCounter == NEW_GAME_MAX) currentState = comp_turn_st; //if the
human has waited enough time, then we go to the comp_turn_st
        break;
    case player_turn_st:
        if(wrongTouch){ //if human touched an already occupied spot
            currentState = waiting_for_touch_st; //go back to waiting state instead of
comp_turn_st
        }
        else if(!minimax_isGameOver(minimax_computeBoardScore(&board, playerIsX))){ //if
the game isn't currently over
            currentState = comp_turn_st; //go to comp_turn_st
            playerIsX = !playerIsX; //flip this boolean so the next player draws the
right symbol
        }
        else if(minimax_isGameOver(minimax_computeBoardScore(&board, playerIsX)))
currentState = game_over_st; //if the game is over, go to game over state
        break;
    case debounce_st:
        currentState = player_turn_st; //no timer needed due to longer period. Just go
immediately to player turn state
        break;
    case comp_turn_st:
        if(!minimax_isGameOver(minimax_computeBoardScore(&board, playerIsX))){ //if the
game isn't over
            currentState = waiting_for_touch_st; //go to waiting state to wait for
player's touch
            newGame = false; //make sure to set this to false so the computer doesn't
keep putting an X in the top left corner
        }
        else currentState = game_over_st; //if the game is indeed over, go to game over
state
        break;
    case waiting_for_touch_st:
        if(display_isTouched()){ //detect touch
```

```
            display_clearOldTouchData(); //clear any former touch data
            currentState = debounce_st; //go to debounce state to make sure it gets
correct touch data
            if(!wrongTouch){ //if the touch wasn't on an already occupied spot
                playerIsX = !playerIsX; //flip boolean since next turn will be the
computer's
            }
            else wrongTouch = false; //if it was an erroneous touch, then reset this
boolean to false until next bad touch comes
        }
        else if(minimax_isGameOver(minimax_computeBoardScore(&board, playerIsX)))
currentState = game_over_st; //otherwise, if the game is over, go to game over state
        break;
    case game_over_st:
        if((buttons_read() & BUTTONS_BTN0_MASK) == BUTTONS_BTN0_MASK){ //only transition
from this state if BTN0 is pressed
            currentState = new_game_st; //go to new game state
            newGame = true; //reset to true since a new game has been started
            playerIsX = true; //reset to true since next turn should be X
            wrongTouch = false; //reset since we assume a touch is fine until it isn't
            newGameCounter = COUNTER_MIN; //reset this wait counter
            for(int r = 0; r < MAX_ROWS; r++){ //iterate through rows
                for(int c = 0; c < MAX_COLUMNS; c++){ //iterate through columns
                    if(board.squares[r][c] == MINIMAX_O_SQUARE) ticTacToeDisplay_drawO(r,
c, true); //if the space has an O, then draw a black O over it
                    else if(board.squares[r][c] == MINIMAX_X_SQUARE)
ticTacToeDisplay_drawX(r, c, true); //if the space has an X, then draw a black X over it
                }
            }
            minimax_initBoard(&board); //re initialize the minimax board
        }
        break;
    default: //need default case for switch statement
        break;
    }

    //state actions
    switch(currentState){
    case new_game_st:
        newGameCounter++; //increment the new game counter
        break;
    case player_turn_st:
        ticTacToeDisplay_touchScreenComputeBoardRowColumn(&row, &column); //get the row
and column based on the touch data
        if(board.squares[row][column] == MINIMAX_EMPTY_SQUARE){ //only take turn there if
the space is empty
            if(playerIsX){ //if the human is X's
                ticTacToeDisplay_drawX(row, column, false); //draw X on screen
                board.squares[row][column] = MINIMAX_X_SQUARE; //mark the corresponding
spot on the minimax board with an X
            }
            else{ //means human is O's
                ticTacToeDisplay_drawO(row, column, false); //draw an O on the screen
                board.squares[row][column] = MINIMAX_O_SQUARE; //mark corresponding spot
on the minimax board with an O
            }
        }
        else{
```

```
            wrongTouch = true; //if the spot touched is not empty, it was an erroneous
touch and we don't want to take the player's turn
        }
        break;
    case comp_turn_st:
        if(newGame){ //if it's the first turn taken by either computer or human
            ticTacToeDisplay_drawX(ROW_0, COL_0, false); //draw an X on the screen in top
left corner
            board.squares[ROW_0][COL_0] = MINIMAX_X_SQUARE; //mark corresponding space on
minimax board
        }
        else{ //it's not the first turn of the game
            minimax_computeNextMove(&board, playerIsX, &row, &column); //call this to
deploy minimax algorithm and get the best move for the computer
            if(playerIsX){ //if computer is X's
                ticTacToeDisplay_drawX(row, column, false); //draw X in picked spot
                board.squares[row][column] = MINIMAX_X_SQUARE; //update minimax board
            }
            else{ //computer is O's
                ticTacToeDisplay_drawO(row, column, false); //draw O in picked spot
                board.squares[row][column] = MINIMAX_O_SQUARE; //update minimax board
            }
        }
        break;
    default: //need default case for switch statement
        break;
    }

}
```