```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;
    }

}
```