Arduino koden

tic_tac_toe_v1.6.7_Final.ino filen

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 40, d5 = 41, d6 = 42, d7 = 43;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

////////////PIN CONFIG///////////
#define PIN 8
#define PZ_PIN 37

#define VRX1 A1
#define VRY1 A2
#define Switch1 2

#define VRX2 A3
#define VRY2 A4
#define Switch2 3
////////////////////////////////////

////////////TONE MACROS////////////
#define ERR_SOUND tone(PZ_PIN, 100, 75);
#define PRESS_SOUND
#define WIN_SOUND
#define DRAW_SOUND
#define PUT_SOUND tone(PZ_PIN, 2200, 100);


//bitSet(TIMSK5, TOIE5);  //sets "TOIEn" in register "TIMSKn" to one.
This bit enables timer overflow interrupts.

//     Check timer 5 registres
/*
 *   Serial.print("TCCR5A:");
 *   Serial.println(TCCR5A);
 *   Serial.print("TCCR5B:");
 *   Serial.println(TCCR5B);
 *   Serial.print("TCCR5C:");
 *   Serial.println(TCCR5C);
 *   Serial.print("OCR5AH:");
 *   Serial.println(OCR5AH);
 *   Serial.print("OCR5AL:");
 *   Serial.println(OCR5AL);
 *   Serial.print("OCR5BH:");
 *   Serial.println(OCR5BH);
 *   Serial.print("OCR5BL:");
 *   Serial.println(OCR5BL);
 */

//ISR for timer overflow:
/*
  ISR(TIMER5_OVF_vect){
    cli();
    OCR3A = sigTable[mode][i]; //PWM on pin 5 uses OCR3A as trigger
value, set it to defind value

    i+=1;
    if (i >= SAMPLESIZE - 1){
       i = 0;
     }//end if
    sei();
  }//end ISR
*/
/////////////SD CARD///////////////
#include <SD.h>
#define SD_PIN 53

//#define DEBUG_writeWinnerToSD //0.53s runtime extra

struct dataStructure {
  char points[3];
  char space = ' ';
  char nameSpace[6];
  char newlineC = '\n';
};



////////////COLOR CONFIG///////////

#define PLAYER_1_COLOR matrix.Color(0,0,255)
```

```
#define PLAYER_2_COLOR matrix.Color(0,255,0)
#define PLAYER_3_COLOR matrix.Color(150,150,150)
#define CURSOR_1_COLOR matrix.Color(255,0,255)
#define CURSOR_2_COLOR matrix.Color(255,255,0)
#define GRID_COLOR matrix.Color(150,150,150)
////////////////////////////////////
#define MENU_DELAY 250
#define GAME_DELAY 200

#define NAME_LENGTH 5
#define menuSize 2
#define SCORE_LENGTH 3
#define LINE_LENGTH (SCORE_LENGTH+NAME_LENGTH+1)
#define MAX_SCORE 999 // This can only be the number of digits defined
by SCORE_LENGTH

#define PLAYER1 false
#define PLAYER2 true

#define NM

#define SLAVE_ADDRESS 0x04

#define RPI_cp 22
//#define DEBUG_sendData
#define DEBUG_timing



///////Macros and variables/////////

#define CURSOR_POS_IN_AREA gameArea[cursorPos[1]][cursorPos[0]]

char gameArea[3][3] = {{ 0,0,0 },
                       { 0,0,0 },
                       { 0,0,0 }};
                       // 1'taller er player 1 og 0 er tom

boolean turn = PLAYER1; //FALSE eller 0 er player 1
boolean inMenu = false; // False if the game is started and 1 if the
statemachine is in the menu

char cursorPos[2] = {0,0};
char Move[2] = {0,0};
char latestPut[2] = {0,0};
char turns = 0;
unsigned long previousMillis = 0;        // will store last time LED was
updated
unsigned int interval = 0;           // interval at which to blink
(milliseconds)

volatile boolean button1_pressed_ISR = false;
volatile boolean button2_pressed_ISR = false;

boolean button1_detected = false;
boolean button2_detected = false;

enum gamestates {start, going, wonP1, wonP2, draw};
gamestates gamestate = start;


enum menustates {start_menu, running_menu, chnm_menu, game_menu};
menustates menustate = start_menu;
menustates previousMenustate = start_menu;
enum players {p1 = 0, p2 = 1};
players player;


char player1Name[NAME_LENGTH] = {'M','A','D','S',' '};
char player2Name[NAME_LENGTH] = {'M','A','R','K',' '};

volatile boolean sendingDataToRPI = false;
boolean doneSending = false;

///////Functions/////////
extern void MoveInArray();
extern bool gameAreaFiled(char xPos, char yPos);
extern bool outOfBounds(char xPos, char yPos);
extern void animate();
extern void drawPiece(char posX, char posY, int color);

//new matrix functions
extern void animateNM();
extern void drawPieceNM(char posX, char posY, int color);
extern void drawGridNM();

extern void detectinput(bool turn);
extern void detectput(bool turn);
extern boolean find_new_pos();
extern void Turndecide();
extern void detectGameCondition();
extern void p1win();
extern void p2win();
extern void drawWin();
extern void rungame();
extern void playerTurnLCD(boolean playerTurn);
extern void resetgame();
extern void inputName(bool chnmPlayer);

//SD card func
extern void serialPrintArr(char* arr, int len);
extern void writeWinnerToSD(boolean winner);

// Debug func
extern void updateD();
extern void printNamesSerial();

// Debug macros
//#define DEBUG_detectGameCondition
//#define DEBUG_Turndecide
//#define DEBUG_MoveInArray
//#define DEBUG_animate
//#define DEBUG_detectinput
//#define DEBUG_find_new_pos
//#define DEBUG_detectput
//#define DEBUG_winner

////////////////////////////////////
// MATRIX DECLARATION:
// Parameter 1 = width of NeoPixel matrix
// Parameter 2 = height of matrix
// Parameter 3 = pin number (most are valid)
// Parameter 4 = matrix layout flags, add together as needed:
//   NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT,
NEO_MATRIX_RIGHT:
//     Position of the FIRST LED in the matrix; pick two, e.g.
//     NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.
//   NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are arranged in
horizontal
//     rows or in vertical columns, respectively; pick one or the other.
//   NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns proceed
//     in the same order, or alternate lines reverse direction; pick
one.
//   See example below for these values in action.
// Parameter 5 = pixel type flags, add together as needed:
```

```
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel
products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels,
not v2)


// Example for NeoPixel Shield.  In this application we'd like to use it
// as a 5x8 tall matrix, with the USB port positioned at the top of the
// Arduino.  When held that way, the first pixel is at the top right,
and
// lines are arranged in columns, progressive order.  The shield uses
// 800 KHz (v2) pixels that expect GRB color data.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(8, 8, PIN,
                                               NEO_MATRIX_TOP     +
NEO_MATRIX_LEFT +
                                               NEO_MATRIX_COLUMNS +
NEO_MATRIX_ZIGZAG,
                                               NEO_GRB            +
NEO_KHZ800);

void setup() {
  // put your setup code here, to run once:
  /////////////////////////////////////
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
  /////////////////////////////
  Serial.begin(9600);
  /////////////////////////////////////////
  while (!Serial) {};
  if (!SD.begin(SD_PIN)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");
  //////////////////////////////////////////


  matrix.begin();
  matrix.setBrightness(40);
  matrix.fillScreen(0);
  matrix.show();

  pinMode(VRX1, INPUT);
  pinMode(VRY1, INPUT);
  pinMode(Switch1, INPUT);
  digitalWrite(Switch1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(Switch1),ISR_button1,FALLING);

  pinMode(VRX2, INPUT);
  pinMode(VRY2, INPUT);
  pinMode(Switch2, INPUT);
  digitalWrite(Switch2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(Switch2),ISR_button2,FALLING);

  Wire.begin(SLAVE_ADDRESS);
  Wire.onRequest(sendData);
  pinMode(RPI_cp, OUTPUT);

}

void loop() {
  unsigned long currentMillis = millis();


  switch(menustate){
      // {start_menu, running_menu, chnm_menu, game_menu};
    case start_menu:
                    static char toggle_menu = 0;
                    inMenu = true; // The ISR of button1 now know that
the player1 is controlling the menu
                    //turn = 0;
                    button1_pressed_ISR = 0;
                    lcd.setCursor(0, 0);
                    lcd.print("Welcome...            ");
                    lcd.setCursor(0,1);
                    lcd.print("Play Tic Tac Toe");
                    toggle_menu = 0; // The menu is now on "Play Tic Tac
Toe"

                    menustate = running_menu; // Go to running_menu
                    break;

    case running_menu:
                      lcd.setCursor(0, 1);
                      detectinput(PLAYER1);

                      //move in menu
                      if(Move[0] > 0){

                        toggle_menu++;      //move right in menu
                        if(toggle_menu == menuSize){toggle_menu=0;};
//wrap around up

                      } else if (Move[0] < 0){

                        toggle_menu--;      //move left in menu
                        if(toggle_menu == -1){toggle_menu =
menuSize-1;}; //wrap around down

                      }//end if move left or right

                      //update menu display if changed
                      if (Move[0] != 0){
                          if(toggle_menu == 0){
                            lcd.print("Play Tic Tac Toe ");
                          }else if(toggle_menu == 1){
                            lcd.print("Change name      ");
                          }
                       } // if Moved

                       //update if the button is pressed
                       if(button1_pressed_ISR){
                          if(toggle_menu == 0){ // Go to Game
                            menustate = game_menu; // Go to the
game_menu next

                          }else if (toggle_menu == 1){ // Go to "Change
name"
                            menustate = chnm_menu;
                          }
                          button1_pressed_ISR = false;
                       }
                    if(menustate != running_menu){
                        inMenu = false; // The Menu StateMachine is now
changing to another menu than "start_menu"
                    }
                    break;

    case chnm_menu:
```

```
                inputName(PLAYER1); // player1
                inputName(PLAYER2); // player2
                menustate = start_menu;
                break;

    case game_menu:
      //{start, going, wonP1, wonP2, draw};
                rungame();

                break; // Break for the game state in the menu SM
  } // Switch/Menu State Machine


  //if menustate chagnges, change delay
  if (previousMenustate != menustate){
    if( menustate ==  running_menu){
      interval = MENU_DELAY ;           //if in menu, apply menu delay
    } else if (menustate == game_menu){
      interval = GAME_DELAY;            //if in game, apply game delay
    }
    previousMenustate = menustate;
  }//end if - previousMenustate

  //while this runtime for the program is less than the inteval
  while(currentMillis - previousMillis <= interval) {
    currentMillis = millis();
    //Serial.println(currnetMillis);

    //Here is the place to add sender code
  }

  currentMillis = millis();             //update the current time, if we
did not enter while loop
  previousMillis = currentMillis;       //update previous time for next
compare

  #ifdef DEBUG_timing
  Serial.print("Ude, det tog:");
  Serial.println(currentMillis - previousMillis);
  #endif



} // void loop
```

```
////////////////////////////////////////////////////
// ISR til knapperne
void ISR_button1(){
   static unsigned long last_interrupt_time = 0;
   unsigned long interrupt_time = millis();
   // If interrupts come faster than 200ms, assume it's a bounce and
ignore
   if (interrupt_time - last_interrupt_time > 200)
   {
      if(turn != true || inMenu){ // If it's the player 1's turn or the
Menu StateMachine is in Menu.
        button1_pressed_ISR = true;
      }
   }
   last_interrupt_time = interrupt_time;
}


void ISR_button2(){
   static unsigned long last_interrupt_time = 0;
   unsigned long interrupt_time = millis();
   // If interrupts come faster than 200ms, assume it's a bounce and
ignore
   if (interrupt_time - last_interrupt_time > 200)
   {
       if(turn){
          button2_pressed_ISR = true;
       }
   }
   last_interrupt_time = interrupt_time;
}
```

LCD_funk.ino filen

```
/* Change name variables with the analog sticks. Only uses
 *  big letters and space.
 *  Call with a turn variabel, false for p1, and true for p2
 *
 */
void inputName(bool chnmPlayer){
  char temp = 0;
  char useOldName_index = 0;
  if(chnmPlayer){//Player 2
    temp = player2Name[useOldName_index];
  }else{//Player 1
    temp = player1Name[useOldName_index];
  }

  bool button_detect;

  bool old_turn;             //hold old global turn value
  old_turn = turn;          //save global turn
  turn = chnmPlayer;             //change global turn value

  chnmPlayer ? button_detect = button2_pressed_ISR : button_detect =
button1_pressed_ISR;

  lcd.clear();
  if (chnmPlayer){
    lcd.print("Player 2 name:");
  } else {
    lcd.print("Player 1 name:");
  }

  lcd.setCursor(0,1);
  lcd.cursor();
  for (char i = 0; i < (NAME_LENGTH); ++i){
    while (!button_detect){
      lcd.setCursor(i,1);
      lcd.write(temp);
      detectinput(chnmPlayer); //detect input for player 1
      delay(250);          //delay for responsive ness feeling

      //increment temp
      if (Move[0] > 0){
        temp++;
      } else if(Move[0] < 0){
        temp--;
      }//increment temp


      //made for looping char choise around
      switch (temp){
        case 91 :
            temp = 32;
            break;
        case 64 :
            temp = 32;
            break;
        case 31 :
            temp = 90;
            break;
        case 33 :
            temp = 65;
            break;
        default :
            break;
      }//end switch

      //poll for button press
      chnmPlayer ? button_detect = button2_pressed_ISR : button_detect =
button1_pressed_ISR;

    }//end while - not pressed

    //save char
    chnmPlayer ? player2Name[i] = temp : player1Name[i] = temp;

    useOldName_index++;
    if(chnmPlayer){//Player 2
      temp = player2Name[useOldName_index];
    }else{//Player 1
      temp = player1Name[useOldName_index];
    }

    //reset press ISR
    button1_pressed_ISR = false;
    button2_pressed_ISR = false;
    button1_detected = false;
    button2_detected = false;
    button_detect = false;
  }//end for - name
 lcd.noCursor();
  turn = old_turn; //restore global turn to previous value

}//end inputName



////////////////////////////////////////////// DEBUG FUNCTION
//////////////////////////////////
void printNamesSerial(){
  for (char i = 0; i < NAME_LENGTH; ++i){
    Serial.print(player1Name[i]);
  }
  Serial.println("");
  for (char i = 0; i < NAME_LENGTH; ++i){
    Serial.print(player2Name[i]);
  }
}
```

RPI_funk.ino filen

```
void sendOneRecord(){

  lcd.setCursor(0 , 1);
  lcd.print("Sending data... ");
  char i = 0;
  digitalWrite(RPI_cp, HIGH);
  while(digitalRead(RPI_cp)){
    //Serial.print("This is value of done sending:");
    //Serial.println(doneSending);
    delay(200);
    if (++i > 15){
      digitalWrite(RPI_cp, LOW);
      i=0;
    }
  }
  digitalWrite(RPI_cp, LOW);
  doneSending = false;
}


void sendData(void) {

  static int index = 0;

#ifdef DEBUG_sendData
  Serial.print("sendData() was called \n");
#endif

  if (turn) {
    Wire.write((byte)player1Name[index]);
  } else {
    Wire.write((byte)player2Name[index]);
  }
  index++;

  //reset index and done sending
  if (index >= NAME_LENGTH) {
    index = 0;
    doneSending = true;
    digitalWrite(RPI_cp, LOW);
  }


#ifdef DEBUG_sendData
  Serial.print("char was sent: ");
  if (turn) {
    Serial.println(player1Name[index]);
  } else {
    Serial.println(player2Name[index]);
  }//end if
#endif
}

/*
  void receiveData(int byteCount){
    while(Wire.available()){
      index = (int)Wire.read();
    Serial.print("index received: ");
    Serial.println(index);
    }
  }
*/
```

SD_func.ino filen

```
 void writeWinnerToSD(boolean winner) {
   File writeFile;                    //for write file
   char score[SCORE_LENGTH];          //to contain score as string
   int finalPoints;                   //to contain score as int
   bool foundName = false;            //used for searching name

   if (sendingDataToRPI) {
           //if sending data, write date to buffer file
```

```
    if (writeFile = SD.open("buffer.txt", O_RDWR)) {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Write file 'buffer.txt' opened");
#endif
    } else {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Error, could not open");
#endif
      return;
    }//end if - open SD
  } else {  //else open score file

    if (writeFile = SD.open("score.txt", O_RDWR)) {
#ifdef DEBUG_writeWinnerToSD
      Serial.println("Write file 'score.txt' opened");
#endif
    } else {

#ifdef DEBUG_writeWinnerToSD
      Serial.println("Error, could not open");
#endif
      return;
    }//end if - open SD
  }//end if - sendingDataToRPI



      //search for player 1 or 2, depending on who wins.
  if (winner) {
    foundName = writeFile.find(player2Name, NAME_LENGTH);
  } else {
    foundName = writeFile.find(player1Name, NAME_LENGTH);
  }//end if - winner find

#ifdef DEBUG_writeWinnerToSD
  Serial.print("The winner is: ");
  serialPrintArr(winner ? player2Name : player1Name, NAME_LENGTH);
#endif


      //if the name is found, 1 is added to players score
      //if not,  a new name has to be added to the list
  if (foundName) {

#ifdef DEBUG_writeWinnerToSD
```

```
    Serial.println("player found in highscore");
    Serial.print("This is pos after find: ");
    Serial.println(writeFile.position());
#endif

    //jump from end of line to begining of line to read score
    writeFile.seek(writeFile.position() - LINE_LENGTH);

#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after seek: ");
    Serial.println(writeFile.position());
#endif

        //read the SCORE_LENGTH char score into char arr
    for (char i = 0; i < SCORE_LENGTH; i++) {
      score[i] = writeFile.read();
    }//end for

#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after read: ");
    Serial.println(writeFile.position());
    serialPrintArr(score, SCORE_LENGTH);
#endif

        //jump cursor back to before read
    writeFile.seek(writeFile.position() - SCORE_LENGTH);

        //make the score into an int and add one
    finalPoints = atoi(score);
    finalPoints++;

    if(finalPoints > MAX_SCORE){ // The score can only be < MAX_SCORE
      finalPoints =  MAX_SCORE;
      Serial.println("The score has been set to MAX_SCORE");
    }

        //write finalPoints formatted to score again
    if(SCORE_LENGTH == 3){
      sprintf(score, "%03d", finalPoints);
    }else{
      Serial.println("The 'SCORE_LENGTH' MACRO has been changed from 3,
go change code in SD_FUNC");
    }
```

```
#ifdef DEBUG_writeWinnerToSD
    Serial.println(finalPoints);
    Serial.print("New score: ");
    serialPrintArr(score, SCORE_LENGTH);
#endif

    //write the score to the place in file
    writeFile.write(score);


  } else {
        //the playername was not found in highscore list

#ifdef DEBUG_writeWinnerToSD
    Serial.println("player not found in highscore");
    Serial.print("This is pos after find:");
    Serial.println(writeFile.position());
#endif
        //position is now at the end of the file, since we could not
find the player


        //now write the first point for player
    for(char i = 0; i < SCORE_LENGTH - 1; i++){
      writeFile.write("0");
    }
    writeFile.write("1 ");

    //writeFile.write("001 ");

        //now write playerName to file
    if (winner) {
      for (char i = 0; i < NAME_LENGTH; i++) {
        writeFile.write(player2Name[i]);
      }
    } else {
      for (char i = 0; i < NAME_LENGTH; i++) {
        writeFile.write(player1Name[i]);
      }
    }//end if - winner find

    writeFile.write("\n");
#ifdef DEBUG_writeWinnerToSD
    Serial.print("This is pos after write:");
    Serial.println(writeFile.position());
```

```
#endif
  }//if else - foundname

      //close the file as the file is only to be used in function
  writeFile.close();
}//end function - writeWinnerToSD



//////////////////////////////// DEBUG FUNCTION
//////////////////////////////////

void serialPrintArr(char* arr, int len) {
  for (char i = 0; i < len; i++) {
    Serial.print(arr[i]);
    Serial.print(", ");
  }
  Serial.println("");
}
```

Tic_tac_toe_funk.ino filen

```
void rungame(){
  switch(gamestate){
        case start:
                  resetgame(); // Restart the game variabels and print
                  gamestate = going;
                  break;
        case going:
                //Print to the LCD
                playerTurnLCD(turn);

                //Get input from player
                detectinput(turn);

                //Move cursor in array
                MoveInArray();

                //Check for button pressed
                detectput(turn);
                //Detect if the game has been won or if its a draw
                detectGameCondition();

                // Decide if the turn should change
                Turndecide();

                // Send out the array to the LED Matrix
                #ifndef NM
                animate();
                #else
                animateNM();
                #endif

                //delay(100);
                break;
        case wonP1...wonP2:
                  delay(1000);
                  matrix.fillScreen(0);
                  matrix.show();

                  // When someone has won, the SM has already shifted
the turn to the other player with the Turndecide() func, so we need to
invert the turn.
                  playerTurnLCD(!turn);

                  if(gamestate == wonP1){
                    boolean toggle = true;

                    for(int j=0;j<3;j++){
                      if(toggle){
                        for(int i=0;i<3;i++){
                          #ifndef NM
                          drawPiece(i,j,PLAYER_1_COLOR);
                          #else
                          drawPieceNM(i,j,PLAYER_1_COLOR);
                          #endif
                          matrix.show();
                          delay(300);
                        }
                        toggle = !toggle;
                      }else{
                        for(int i=2;i>-1;i--){
                          #ifndef NM
                           drawPiece(i,j,PLAYER_1_COLOR);
                          #else
```

```
                         drawPieceNM(i,j,PLAYER_1_COLOR);
                        #endif
                         matrix.show();
                         delay(300);
                      }
                      toggle = !toggle;
                    }
                  }
                }else if(gamestate == wonP2) {
                  boolean toggle = true;

                  for(int j=0;j<3;j++){
                    if(toggle){
                      for(int i=0;i<3;i++){
                        #ifndef NM
                        drawPiece(i,j,PLAYER_2_COLOR);
                        #else
                        drawPieceNM(i,j,PLAYER_2_COLOR);
                        #endif
                        matrix.show();
                        delay(300);
                      }
                      toggle = !toggle;
                    }else{
                      for(int i=2;i>-1;i--){
                        #ifndef NM
                        drawPiece(i,j,PLAYER_2_COLOR);
                        #else
                        drawPieceNM(i,j,PLAYER_2_COLOR);
                        #endif
                        matrix.show();
                        delay(300);
                      }
                      toggle = !toggle;
                    }
                  }
                }
                // When someone has won, the SM has already shifted
the turn to the other player with the Turndecide() func, so we need to
invert the turn to get the winner.
                writeWinnerToSD(!turn);
                //send til RPI
                sendOneRecord();
                gamestate = start;
                menustate = start_menu;
```

```
                  break;

           //The game has ended in a draw
           case draw:
                    delay(1000);
                    matrix.fillScreen(0);
                    matrix.show();
                    lcd.setCursor(0,1);
                    lcd.print("It's a draw!");

                    boolean toggle = true;

                    for(int j=0;j<3;j++){
                      if(toggle){
                          for(int i=0;i<3;i++){
                            #ifndef NM
                            drawPiece(i,j,PLAYER_3_COLOR);
                            #else
                            drawPieceNM(i,j,PLAYER_3_COLOR);
                            #endif
                            matrix.show();
                            delay(300);
                          }
                          toggle = !toggle;
                      }else{
                          for(int i=2;i>-1;i--){
                            #ifndef NM
                             drawPiece(i,j,PLAYER_3_COLOR);
                             #else
                             drawPieceNM(i,j,PLAYER_3_COLOR);
                             #endif
                             matrix.show();
                             delay(300);
                          }
                          toggle = !toggle;
                      }
                    }
                    gamestate = start;
                    menustate = start_menu;
                    break;
              } // Switch/ Game State Machine
}//end run-game

/*
 * Decides whos turn it is
 */
void Turndecide(){
    if(turn){// Player 2
        if(button2_detected ){
          #ifdef DEBUG_Turndecide
              Serial.println("Its now player 1's turn");
          #endif
          turn = PLAYER1;
          button2_detected = false;
        }
    }else{// Player 1
        if(button1_detected ){
          #ifdef DEBUG_Turndecide
              Serial.println("Its now player 2's turn");
          #endif
          turn = PLAYER2;
          button1_detected = false;
        }
    }
}

/*
 *  Function takes the move, sees if it is possible and if so, moves the
cursor
 */
void MoveInArray(){
  char newCursorPos[2] = {0,0}; // for checking
  newCursorPos[0] = cursorPos[0] + Move[0];
  newCursorPos[1] = cursorPos[1] + Move[1];

  if (outOfBounds(newCursorPos[0],newCursorPos[1])){
    #ifdef DEBUG_MoveInArray
        Serial.println("First bound check failed");
    #endif

    return;
  }

  cursorPos[0] = newCursorPos[0];
  cursorPos[1] = newCursorPos[1];
  return;
}

/*
 * returns true if the input position is out of the array
 */
bool outOfBounds(char xPos, char yPos){
  if (xPos > 2 || xPos < 0 || yPos > 2 || yPos < 0){
    return true;
  } else {
    return false;
  }
}

/*
 * returns true if the input position is filled already
 */
bool gameAreaFiled(char xPos, char yPos){
  if ((gameArea[yPos] [xPos]) != 0){
    return true;
  } else {
    return false;
  }
}

/* NEW LED MATRIX FUNC
 * Runs the animation funcktion, animates the board and cursor on the
new led matrix
 */
void animateNM(){
  matrix.fillScreen(0);

  drawGridNM(); //draw grid

  //run through game area and print it in players colors
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      switch (gameArea[j][i]){
        case 1:
          #ifdef DEBUG_animate
              Serial.println("Player 1's tile is being printed");
          #endif
          drawPieceNM(i, j,PLAYER_1_COLOR);
          break;
        case 2:
          #ifdef DEBUG_animate
              Serial.println("Player 2's tile is being printed");
          #endif
          drawPieceNM(i, j,PLAYER_2_COLOR);
          break;
        default:
          break;
      }//end switch
    }//end for
  }//end for

  //animate cursor position
  if(gamestate == going){
      #ifdef DEBUG_animate
            Serial.println("The Cursor is being printed");
      #endif
      drawPieceNM(cursorPos[0], cursorPos[1], (turn ? CURSOR_2_COLOR :
CURSOR_1_COLOR));
  }
  matrix.show();
}

/* NEW LED MATRIX FUNC
 * Function draws a piece on the NEW LED matrix from a gameArea
coordinate
 */
void drawPieceNM(char posX, char posY, int color){
  matrix.drawPixel((posX * 3),(posY * 3),color);
  matrix.drawPixel((posX * 3)+1,(posY * 3),color);
  matrix.drawPixel((posX * 3),(posY * 3)+1,color);
  matrix.drawPixel((posX * 3)+1,(posY * 3)+1,color);
}
extern void drawGridNM(){
  for (char y = 2; y < 6; y += 3){
    for (char x = 0; x < 8; x++){
      matrix.drawPixel(x, y, GRID_COLOR);
    }
  }
  for (char x = 2; x < 6; x += 3){
    for (char y = 0; y < 8; y++){
      matrix.drawPixel(x, y, GRID_COLOR);
    }
  }

}
 /*Runs the animation funcktion, animates the board and cursor
  */
void animate(){
  matrix.fillScreen(0);
  //run through game area and print it in players colors
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      switch (gameArea[j][i]){
        case 1:
          #ifdef DEBUG_animate
              Serial.println("Player 1's tile is being printed");
          #endif
          drawPiece(i, j,PLAYER_1_COLOR);
          break;
        case 2:
          #ifdef DEBUG_animate
              Serial.println("Player 2's tile is being printed");
          #endif
          drawPiece(i, j,PLAYER_2_COLOR);
          break;
        default:
          break;
      }//end switch
    }//end for
  }//end for

  //animate cursor position
  if(gamestate == going){
      #ifdef DEBUG_animate
            Serial.println("The Cursor is being printed");
      #endif
      drawPiece(cursorPos[0], cursorPos[1], (turn ? CURSOR_2_COLOR :
CURSOR_1_COLOR));
  }
  matrix.show();
}

/*Function draws a piece on the LED matrix from a gameArea coordinate
 */
void drawPiece(char posX, char posY, int color){
  matrix.drawPixel((posX * 2),(posY*2),color);
  matrix.drawPixel((posX * 2)+1,(posY*2),color);
  matrix.drawPixel((posX * 2),(posY*2)+1,color);
  matrix.drawPixel((posX * 2)+1,(posY*2)+1,color);
}

/* DetectInput() recieves a boolean "turn" which indicades who's turn it
is. The func will change the Move[] array and return */
void detectinput(bool turn){
  int x,y = 0;
```

```
if(turn){ // Chooses which analog stick that needs to be read from
    // Player 2
    x = analogRead(VRX2); // Read from the x-axis
    y = analogRead(VRY2); // Read from the y-axis

    #ifdef DEBUG_detectinput
        Serial.print("VRX2:");
        Serial.print(x);
        Serial.print("\tVRY2:");
        Serial.println(y);
    #endif

}else{
    // Player 1
    x = analogRead(VRX1); // Read from the x-axis
    y = analogRead(VRY1); // Read from the y-axis

    #ifdef DEBUG_detectinput
        Serial.print("VRX1:");
        Serial.print(x);
        Serial.print("\tVRY1:");
        Serial.println(y);
    #endif
}

//Decides which direction the analog stick is facing on the x-axis
if( x > 800){
    #ifdef DEBUG_detectinput
        Serial.println("The analog stick is facing left");
    #endif

    Move[0] = 1;
}else if(x < 300){
    #ifdef DEBUG_detectinput
        Serial.println("The analog stick is facing right");
    #endif
    Move[0] = -1;
}else{
    Move[0] = 0;
}

  //Decides which direction the analog stick is facing on the y-axis
if( y > 800){
    #ifdef DEBUG_detectinput
        Serial.println("The analog stick is facing up");
        #endif
        Move[1] = 1;
    }else if(y < 300){
        #ifdef DEBUG_detectinput
            Serial.println("The analog stick is facing down");
        #endif
        Move[1] = -1;

    }else{
        Move[1] = 0;
    }
    return;
}

/*The function finds the first "legal" position in the gameArea to place
the cursor*/
boolean find_new_pos(){
        for(char i = 0;i<3;i++){
              for(char j = 0;j<3;j++){
                  if(gameAreaFiled(j, i) == 0){
                      cursorPos[0] = j;
                      cursorPos[1] = i;
                            #ifdef DEBUG_find_new_pos
                                Serial.println("A new position has been
found");
                            #endif

                      return 1;
                  }
              }
        }
      return 0; // No new position available
}

/*The function detects a button press, inserts a '1' or '2'
 *on the position of the cursor in the gameArea Array and
 *finds a new legal position for the cursor */
void detectput(bool turn){

    if(turn){
        if(button2_pressed_ISR){
              #ifdef DEBUG_detectput
                   Serial.println("Button 2 has been pressed");
              #endif
              button2_pressed_ISR = false; // These var must be reset,
because they can be toggled anytime by the ISR
              button1_pressed_ISR = false;

              if (!gameAreaFiled(cursorPos[0],cursorPos[1] )){
                  PUT_SOUND
                  //Serial.println("second filld check failed");
                  button2_detected = true; // This var must be set,
because we need to know that the game has registered a button press
                  turns++; // Increment the amount of turns
                  CURSOR_POS_IN_AREA = 2; // Indsætter player 2's træk
på banen
                  //Saves the cursorposition where a 1 or 2 has been
placed. This is used in the func "detectGameCondition()"
                  latestPut[0] = cursorPos[0];
                  latestPut[1] = cursorPos[1];
                  #ifdef DEBUG_detectput    // Prints out the array on
the Serial monitor
                    updateD();
                  #endif
                  find_new_pos(); //Find a new "legal" position for the
cursor

                  return;
              }  else {
                ERR_SOUND
              }

        } // end if
  }else{
         if(button1_pressed_ISR){
              #ifdef DEBUG_detectput
                   Serial.println("Button 1 has been pressed");
              #endif
              button1_pressed_ISR = false;// These var must be reset,
because they can be toggled anytime by the ISR
              button2_pressed_ISR = false;

              if (!gameAreaFiled(cursorPos[0],cursorPos[1] )){
                PUT_SOUND
                button1_detected = true;  //This var must be set,
because we need to know that the game has registered a button press
                turns++; // Increment the amount of turns
                CURSOR_POS_IN_AREA = 1; // Indsætter player 1's træk på
banen
                //Saves the cursorposition where a 1 or 2 has been
placed. This is used in the func "detectGameCondition()"
                latestPut[0] = cursorPos[0];
                latestPut[1] = cursorPos[1];
                #ifdef DEBUG_detectput
                  updateD();
                #endif
                find_new_pos(); //Find a new "legal" position for the
cursor
              } else {
                ERR_SOUND
              }// end if gameAreaField()
          } // end if
  } // end else
} // end func

/*
 * Make life great again
 * Detect the winner/draw and change game state.
 */
void detectGameCondition(){
  char xCheck = 0;
  char yCheck = 0;
  if (button1_detected || button2_detected){

    #ifdef DEBUG_detectGameCondition
        Serial.println("Check win begun");
    #endif
/////////////////// Check column win ///////////////////
    //first make x coordinate to check
    xCheck = latestPut[0] + 1;
    if (xCheck > 2){
      xCheck = 0;
    }//end if - xCheck overflow

    #ifdef DEBUG_detectGameCondition
        Serial.print("First X coordinate generatred:\t");
        Serial.println((int)xCheck);
    #endif

    if (gameArea[latestPut[1]][xCheck] ==
gameArea[latestPut[1]][latestPut[0]]){

      #ifdef DEBUG_detectGameCondition
         Serial.println("First test coordnate matches latest put");
      #endif
      //second make x coordinate to check
      xCheck++;
      if (xCheck > 2){
        xCheck = 0;
      }//end if - xCheck overflow

      #ifdef DEBUG_detectGameCondition
          Serial.print("Second X coordinate generatred:\t");
          Serial.println((int)xCheck);
      #endif

      if (gameArea[latestPut[1]][xCheck] ==
gameArea[latestPut[1]][latestPut[0]]){

        #ifdef DEBUG_detectGameCondition
           Serial.println("Someone won");
        #endif
        if (gameArea[latestPut[1]][latestPut[0]] == 1){
          p1win();
          return;

        } else if (gameArea[latestPut[1]][latestPut[0]] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - second check "win check"
    }//end if - first check

////////////////// Check row win //////////////////
    //first make y coordinate to check
    yCheck = latestPut[1] + 1;
    if (yCheck > 2){
      yCheck = 0;
    }//end if - yCheck overflow

    #ifdef DEBUG_detectGameCondition
        Serial.print("First Y coordinate generatred:\t");
        Serial.println((int)yCheck);
    #endif

    if (gameArea[yCheck][latestPut[0]] ==
gameArea[latestPut[1]][latestPut[0]]){

      #ifdef DEBUG_detectGameCondition
```

```
        Serial.println("First test coordnate matches latest put");
      #endif

      //second make x coordinate to check
      yCheck++;
      if (yCheck > 2){
        yCheck = 0;
      }//end if - yCheck overflow

      #ifdef DEBUG_detectGameCondition
          Serial.print("Second Y coordinate generatred:\t");
          Serial.println((int)yCheck);
      #endif

      if (gameArea[yCheck][latestPut[0]] ==
gameArea[latestPut[1]][latestPut[0]]){

        #ifdef DEBUG_detectGameCondition
           Serial.println("Someone won");
        #endif

        if (gameArea[latestPut[1]][latestPut[0]] == 1){
          p1win();
          return;

        } else if (gameArea[latestPut[1]][latestPut[0]] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - second check "win check"
    }//end if - first check

////////////////// Check diagonal win //////////////////
    if ((latestPut[0] == latestPut[1]) || ((latestPut[0] - latestPut[1])
% 2 == 0)){
      //check if latest put is inside diagonals
      //might be more work to check if i should check, please reasearch
that! -Mads

      if ((gameArea[0][0] == gameArea[1][1] && gameArea[1][1] ==
gameArea[2][2]) || (gameArea[0][2] == gameArea[2][0] && gameArea[1][1]
== gameArea[2][0])){

        #ifdef DEBUG_detectGameCondition
            Serial.println("Diagonal win detected");
        #endif
        //winner clearly has the middel
        if (gameArea[1][1] == 1){
          p1win();
          return;

        } else if (gameArea[1][1] == 2){
          p2win();
          return;
        }//end if - change gamestate
      }//end if - diagonal win
    }//end if - last put on diagonal
      if(turns == 9){
        drawWin();
        return;
      }
  }//end if buttons detected
}//end function detect win state



/* Printning out the Array on the Serial monitor (DEBUG)*/
void updateD(){
  for(size_t i = 0; i < 3; i++){
    for (size_t j = 0; j < 3; j++){
      Serial.print((int)gameArea[i][j]);
    }
    Serial.println();
  }Serial.println();

}//end updateD

void p1win(){
  gamestate = wonP1;
  #ifdef DEBUG_winner
       Serial.println("Player 1 won");
  #endif

}//end p1win

void p2win(){
  gamestate = wonP2;
  #ifdef DEBUG_winner
       Serial.println("Player 2 won");
  #endif
```

```
}//end p2win

void drawWin(){
  gamestate = draw;
  #ifdef DEBUG_winner
         Serial.println("draw detected, 9 moves used");
  #endif

}//end drawWin


void playerTurnLCD(boolean playerTurn){

   if(playerTurn){ // Player 2
      if(gamestate == going){
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
              lcd.print(player2Name[i]);
          }
      }else if(gamestate == wonP2){
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
              lcd.print(player2Name[i]);
          }
          lcd.print(" wins");
      }
   }else{ // Player 1
      if(gamestate == going){ // Prints the player name for whome has
the turn
          lcd.setCursor(0,1);
          for(char i = 0; i < NAME_LENGTH;i++){
              lcd.print(player1Name[i]);
          }
      }else if(gamestate == wonP1){ // Prints the player name for who
won.
          lcd.setCursor(0,1);
          for(char i = 0;i<NAME_LENGTH;i++){
              lcd.print(player1Name[i]);
          }
          lcd.print(" wins");
      }
   }
   lcd.noCursor(); // Remove the cursor
}
```

```
void resetgame(){

                  button1_pressed_ISR = false;
                  button2_pressed_ISR = false;
                  button1_detected = false;
                  button2_detected = false;
                  cursorPos[0] = 0;
                  cursorPos[1] = 0;
                  latestPut[0]= 0;
                  latestPut[1]= 0;
                  turns = 0;

                  //Write to LCD
                  lcd.clear();
                  lcd.setCursor(0, 0);
                  lcd.print("Tic Tac Toe");
                  //Reset the gameArea and the LED Maxtrix
                  for(int i = 0;i<3;i++){
                      for(int j = 0;j<3;j++){
                        gameArea[i][j] = 0;
                      }
                  }
                  matrix.fillScreen(0);
                  matrix.show();
}
```