

# Tic Tac Toe mit 2,4“ Touchdisplay

Als Hobbybastler und Vater will man nicht immer nur Nützliches mit seinem Elektronikequipment bauen, sondern will auch ein bisschen Spaß haben bzw. die staunenden Gesichter der Kinder sehen. Warum also nicht das Hobby mal mit Spaß verbinden und eine Runde Tic Tac Toe gegen einen Arduino spielen? Damit kann man sich und die Kinder dann eine ganze Weile beschäftigen. Mehr als ein 2,4“ TFT LCD Touchdisplay und ein Arduino Uno mit Stromversorgung ist dafür nicht nötig.

## Der Ablauf

Die Spielregeln sind relativ einfach, eine Reihe von drei X- oder O-Zeichen verhelfen einem zum Sieg, interessanter jedoch ist, wie das Programm später funktionieren soll. Beim ersten Start der Software werden Sie mit dem Startscreen, siehe Abbildung 1, begrüßt.



Abbildung 1: Startscreen Tic Tac Toe

Sie haben dann die Wahl zwischen einem einfachen „Easy“ oder einem schweren „Hard“ Spiel gegen den Arduino. Bei dem einfachen Spiel starten Sie mit dem ersten Zug und der Arduino wählt zufällig aus, wo sein nächster Zug sein wird. Primär geht es im einfachen Spiel dem Arduino nicht darum, zu gewinnen oder Sie am Sieg zu hindern.

Im schweren Spiel hingegen darf der Arduino beginnen und wird versuchen, sie vom gewinnen abzuhalten und gleichzeitig versuchen, so schnell wie möglich selbst zu gewinnen. Der Knackpunkt beim ersten Zug vom Arduino in der schweren Variante besteht darin, dass nicht immer die Mitte, was einen strategischen Vorteil bietet, sondern ab und zu auch mal eine zufällige Position ausgewählt wird.

Nach jedem Zug prüft der Arduino, ob es schon einen Sieger gibt oder die möglichen neun Züge verspielt wurden. Ist das Spiel vorbei, wird geprüft, wer gewonnen hat und der entsprechende EndScreen wird angezeigt. Um Schummeln zu vermeiden, kann immer nur ein leeres Feld ausgewählt werden, wie es auch bei der Papiervariante der Fall ist.

## Benötigte Hardware

Die Hardware für diesen kleinen Zeitvertreib können Sie bei AZ-Delivery oder aber bei Amazon bestellen.

Pos	Anzahl	Bauteil	Link
1	1	Arduino Uno	<a href="https://az-delivery.com">https://az-delivery.com</a> <a href="https://www.amazon.de">https://www.amazon.de</a>
2	1	2,4" TFT LCD Touchdisplay	<a href="https://az-delivery.com">https://az-delivery.com</a> <a href="https://www.amazon.de">https://www.amazon.de</a>

*Tabelle 1: Benötigte Hardware für das Projekt*

Generell können Sie auch ein anderes Touchdisplay benutzen, jedoch müssen Sie dann auf das Pinout des eingesetzten Touchdisplays achten und ggf. den Quellcode modifizieren.

Bevor Sie anfangen, stecken Sie das 2.4" TFT LCD Touchdisplay gemäß dem eBook auf den Arduino Uno.

## Benötigte Software

Die benötigte Software für dieses Projekt ist auch überschaubar:

- Arduino IDE (<https://www.arduino.cc/en/Main/Software>), hier am besten die aktuellste Version herunterladen
- Die Bibliothek **MCUFriend\_kbv** mit allen Abhängigkeiten, die Sie über die Bibliotheksverwaltung, siehe <https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/arduino-ide-programmieren-fuer-einsteiger-teil-1> Abschnitt Bibliotheksverwaltung, installieren müssen.

## Kalibrierungsdaten vom Display ermitteln

Bevor Sie nun den Quellcode für das Spiel übernehmen, müssen Sie noch eine Displaykalibrierung durchführen, damit die Positionsangaben beim Berühren vom Display auch korrekt sind. Damit Sie nun nicht lange suchen müssen, hat der Autor der **MCUFriend\_kbv**-Bibliothek ein entsprechendes Beispiel mit dem Namen „TouchScreen\_Calibr\_native“ bereitgestellt, siehe Abbildung 2.

Wichtig ist, dass Sie bitte den Seriellen Monitor der Arduino IDE starten, da Sie hier Werte kopieren müssen.

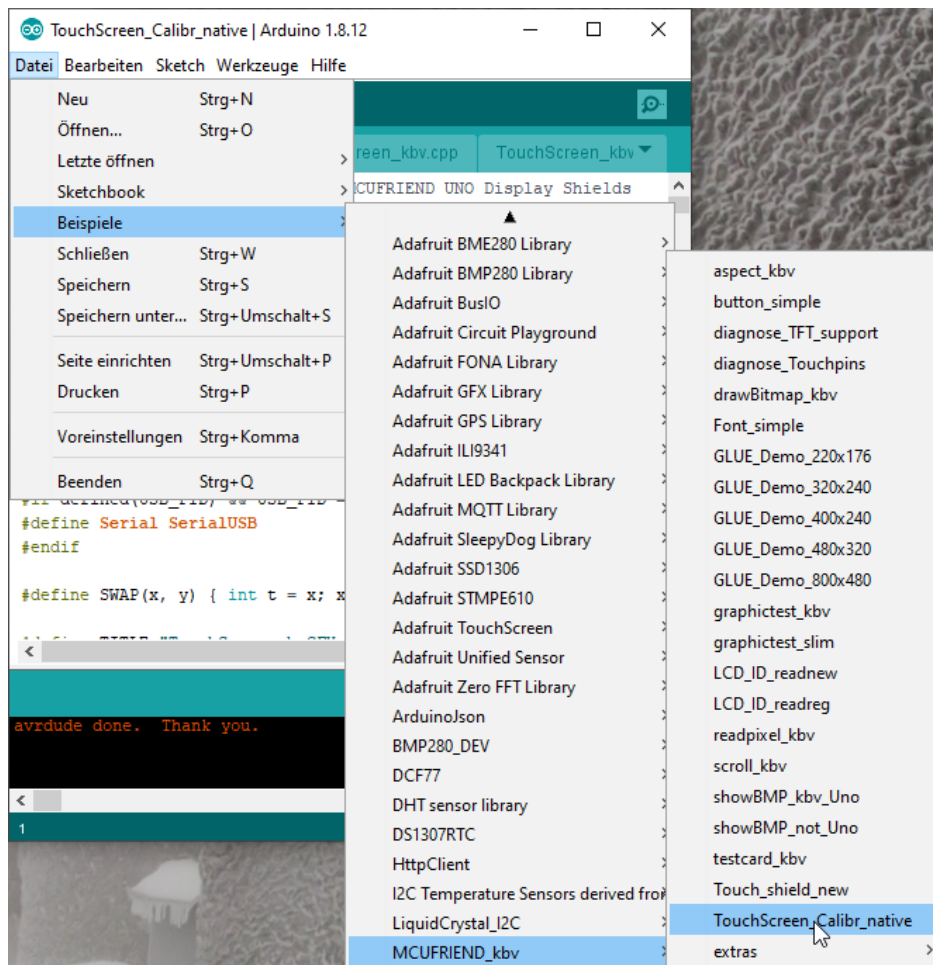


Abbildung 2: Arduino Beispiel Touch-Kalibrierung öffnen

Folgen Sie den Anweisungen auf dem Touchdisplay, drücken und halten Sie die angezeigten Positionsmarker, die weiß markiert sind. Haben Sie alle Positionsmarken durch, wird Ihnen auf dem Touchdisplay die Kalibrierung vom Display ausgegeben, siehe Abbildung 3. Hier wird unterschieden zwischen Hochkant „Portrait“ und Querformat „Landscape“.

Für das Spiel brauchen Sie die Daten für die seitliche „Landscape“ Ausrichtung.

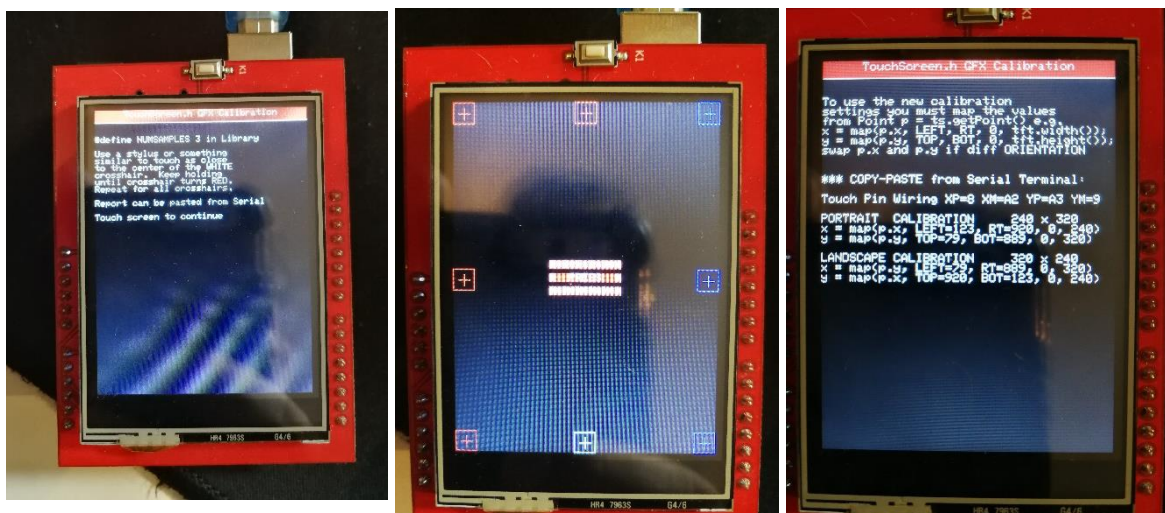
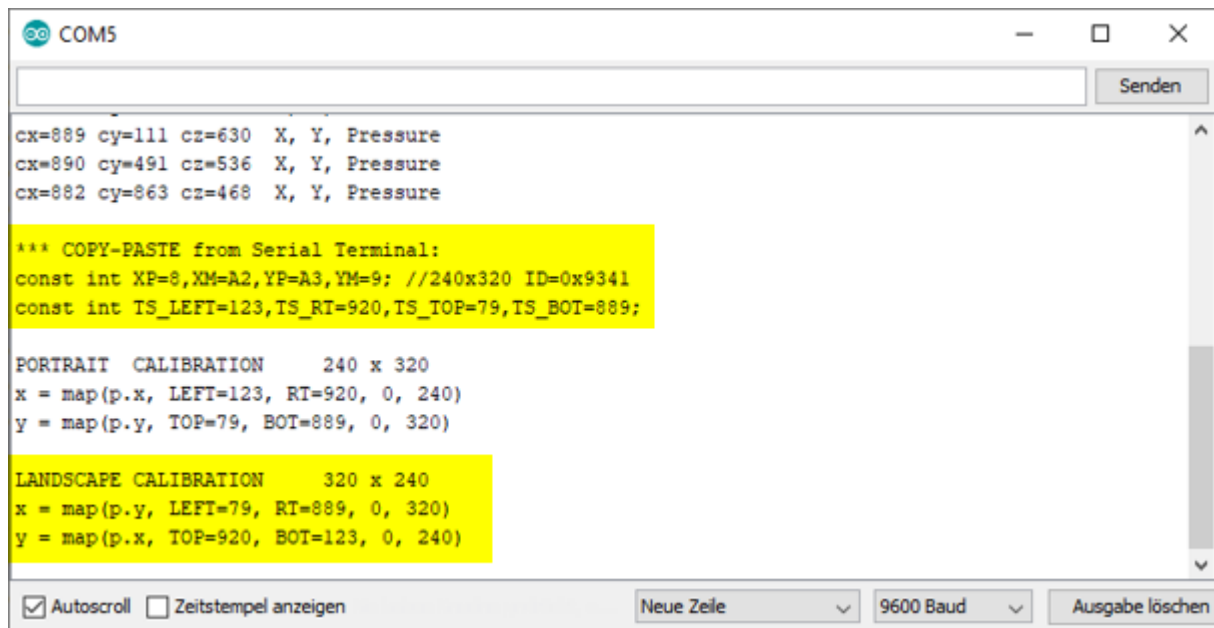


Abbildung 3: Kalibrierung Touchdisplay durchführen

Diese Werte müssen Sie nun nicht abschreiben, sondern können die benötigten Werte aus dem Seriellen Monitor kopieren, siehe Abbildung 4.

The screenshot shows the 'COM5' Serial Monitor window. The text inside is as follows:

```
cx=889 cy=111 cz=630 X, Y, Pressure
cx=890 cy=491 cz=536 X, Y, Pressure
cx=882 cy=863 cz=468 X, Y, Pressure

*** COPY-PASTE from Serial Terminal:
const int XP=8, XM=A2, YP=A3, YM=9; //240x320 ID=0x9341
const int TS_LEFT=123, TS_RT=920, TS_TOP=79, TS_BOT=889;

PORTRAIT CALIBRATION    240 x 320
x = map(p.x, LEFT=123, RT=920, 0, 240)
y = map(p.y, TOP=79, BOT=889, 0, 320)

LANDSCAPE CALIBRATION   320 x 240
x = map(p.y, LEFT=79, RT=889, 0, 320)
y = map(p.x, TOP=920, BOT=123, 0, 240)
```

The bottom of the window has controls: ☒ Autoscroll, ☐ Zeitstempel anzeigen, a dropdown for 'Neue Zeile', a dropdown for '9600 Baud', and a button 'Ausgabe löschen'.

Abbildung 4: Kalibrierungsausgabe im Seriellen Monitor

Wichtig dabei sind die beiden Zeilen unter „\*\*\* COPY\_PASTE from Serial Terminal“ und die beiden Zeilen unter „LANDSCAPE CALIBRATION“. Die ersten Werte sind das Pinout, um das Touchdisplay anzusteuern, die unteren Werte das Mapping, um die korrekte Pixelposition vom Display zu erhalten.

## Was ist Mapping?

Bevor Sie lange googlen, was „Mapping“ bedeutet, möchte ich Ihnen diese Frage gerne anhand von unserem Touchdisplay erläutern. Sie erhalten beim Drücken auf das Touchdisplay Informationsdaten über die Position und den Druck. Diese Werte spiegeln aber nicht die korrekten Pixelkoordinaten des Displays wider, sondern die Widerstandswerte vom Display. Genau an dieser Stelle kommt das Mapping zum Einsatz. Die ermittelten Grenzwerte werden genutzt, um die X- und Y-Pixelkoordinaten vom Display zu errechnen. Die genaue Funktion von map können Sie auch in der Referenz von Arduino unter <https://www.arduino.cc/reference/de/language/functions/math/map/> nachlesen.

## Anpassung im Quellcode

Wenn Sie den Quellcode vom Spiel öffnen, finden Sie in Zeile 19 den Kommentar „// COPY-PASTE from Serial Terminal TouchScreen\_Calibr\_native“. Die beiden Zeilen dahinter müssen nun von Ihnen durch Ihre kopierten Werte hinter „\*\*\* COPY\_PASTE from Serial Terminal“ der Kalibrierung ersetzt werden. Damit wird das Touchdisplay korrekt angesteuert. Sollten Sie das oben genannte Touchdisplay nutzen, so müssen Sie an dieser Stelle erst einmal nichts verändern müssen.

In der Funktion bool Touch\_getXY(void), siehe Zeile 38 im Quellcode, finden Sie den Kommentar „//Update mapping from calibration“. Direkt danach sehen Sie das Mapping für die X- und Y-Koordinate vom Touchdisplay. Ersetzen Sie an dieser Stelle bitte die vorhandenen Werte mit Ihren kopierten Werten vom Seriellen Monitor der „LANDSCAPE



CALIBRATION“, wobei Sie nur die Werte für LEFT, RT, TOP, BOT innerhalb der Klammern ersetzen sollten und nicht den gesamten Text, siehe Abbildung 5.

```
//Returns if there is an touch-action on screen
//Overwrite X and y coordinate and set boolflag
bool Touch_getXY(void)
{
    TSPoint p = ts.getPoint();
    pinMode(YP, OUTPUT);      //restore shared pins
    pinMode(XM, OUTPUT);
    digitalWrite(YP, HIGH);   //because TFT control pins
    digitalWrite(XM, HIGH);
    bool pressed = (p.z > MINPRESSURE && p.z < MAXPRESSURE);
    if (pressed)
    { //Update mapping from calibration
        pixel_x = map(p.y, 88, 899, 0, 320); //kbv makes sense to me
        pixel_y = map(p.x, 917, 122, 0, 240);
    }
    return pressed;
}
```

Abbildung 5: Kalibrierungsdaten für Mapping übernehmen

## Der Quellcode

Da der Quellcode über 600 Zeichen hat und Sie zusätzlich noch eine weitere Datei für das Generieren vom X- und O- Symbol benötigen, soll an dieser Stelle nur der Ablauf erläutert werden. Den Quellcode können sie unter <https://github.com/M3taKn1ght/Blog-Repo/tree/master/TicTacToe> downloaden.

Am Anfang vom Quellcode wird ein Objekt der Klasse TouchScreen erzeugt, siehe Zeile 22 im Quellcode. An dieser Stelle werden die Pins für Kommunikation und der Erfassung der analogen Werte vom Touchdisplay übergeben.

In der Setup-Funktion werden der Serielle Monitor und das Display gestartet, gleichzeitig aber auch der Startscreen samt Buttons erzeugt. Interessant sind dabei die Zeilen 86 und 88, in denen das Aussehen der beiden Buttons vergeben wird, siehe Abbildung 6.

```
btnEasy.initButton(stft, 100, 190, 100, 40, GREEN, BLUE, GREEN, "Easy", 2);
btnEasy.drawButton();
btnHard.initButton(stft, 220, 190, 100, 40, YELLOW, RED, YELLOW, "Hard", 2);
btnHard.drawButton();
```

Abbildung 6: Initialisierung der beiden Buttons

Die drei angegebenen Farben definieren sich wie folgt. Die erste Farbe ist die Rahmenfarbe, die zweite Farbe ist die Füllfarbe und die letzte Farbe ist die Textfarbe. Hier können Sie gerne ein eigenes Design kreieren.

In der loop-Funktion wird überwacht, ob der Spieler eine Eingabe auf dem Display macht und an welcher Position. Dabei wird die Funktion Touch\_getXY() aufgerufen, welche die entsprechenden Koordinaten über das Mapping ermittelt und in die globalen Variablen pixel\_x und pixel\_y schreibt, siehe Code 1.

```
//Check in "main-menu" if human select a game
void loop() {
    unsigned long currentMillis = millis();
    if(currentMillis - previousMillis >= interval )
    {
```

```

previousMillis = currentMillis;
if(iEnableButtons)
{
    bPressed = Touch_getXY();
    if(bPressed)
        Serial.println("X: " + String(pixel_x) + " Y: " + String(pixel_y));
    btnEasy.press(bPressed && btnEasy.contains(pixel_x,pixel_y));
    if (btnEasy.justPressed()){
        //btnEasy.drawButton(true);
        Serial.println("Easy game");
        ResetGame();
        playGame(false); //Easy Mode
    }
    btnHard.press(bPressed && btnHard.contains(pixel_x,pixel_y));
    if (btnHard.justPressed()){
        //btnHard.drawButton(true);
        Serial.println("Hard game");
        ResetGame();
        playGame(true); //Hard Mode
    }
}
}
}

```

*Code 1: Loop-Funktion des Programms*

Sollte die Eingabe auf einem der beiden Buttons „Easy“ und „Hard“ sein, so werden das Spielfeld über die Funktion ResetGame() generiert und die beiden Buttons für einfaches oder schwieriges Spiel deaktiviert, siehe Abbildung 7.

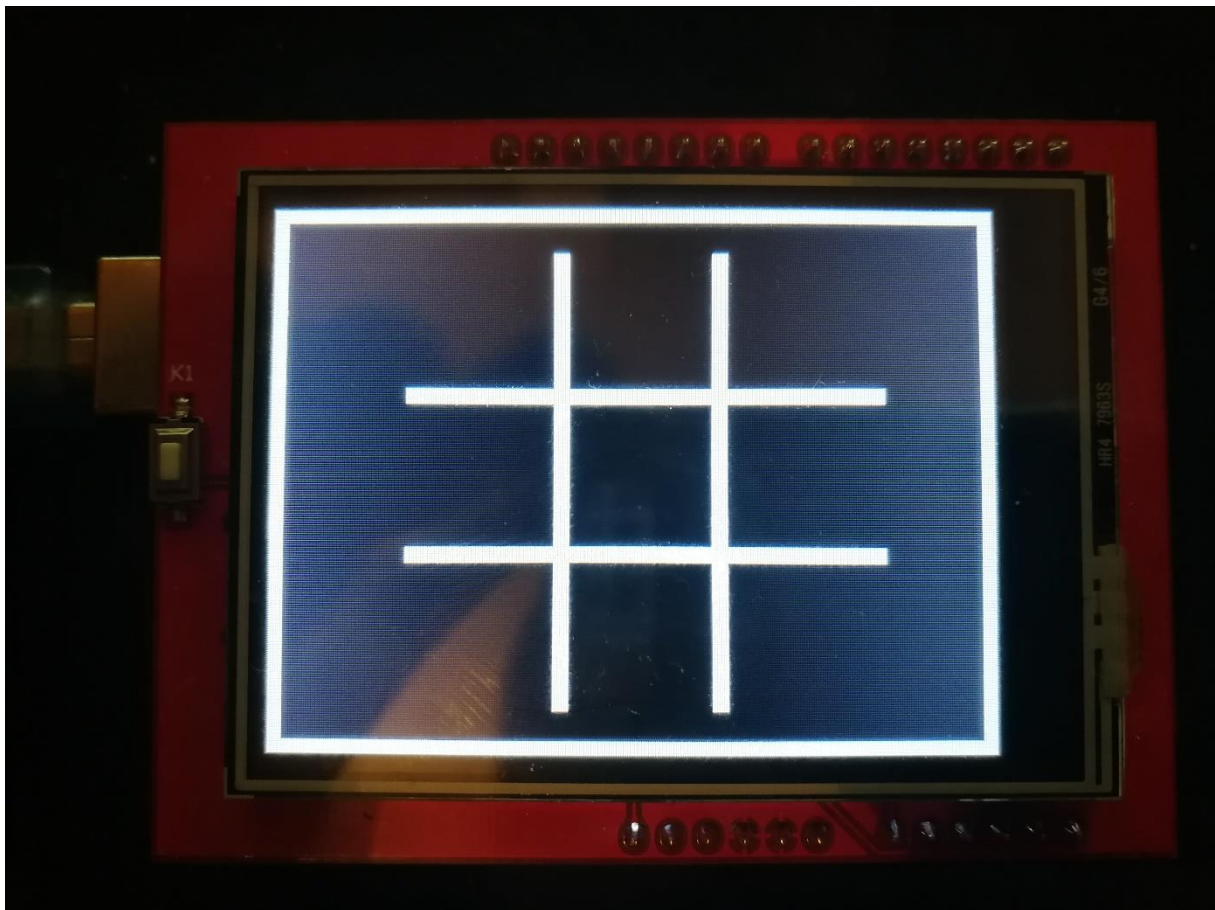


Abbildung 7: Spielfeld im simplen weiß

Auch hier haben Sie die Möglichkeit, in der Funktion `ResetGame()` dem Rahmen und allen vertikalen und horizontalen Linien eine eigene Farbe zu geben. Wie wäre es denn z.B. mit einem bunten Spielfeld, siehe Abbildung 8?

```
//Generate play-screen and reset all vars
//Each element from field can have own color
void ResetGame(){
    iEnableButtons = 0;
    int iCnt = 0;
    for(iCnt = 0; iCnt < 9; iCnt++)
        iSetMark[iCnt] = 0;
    iWinner = 0; //Nobody wins so far :)
    iMoves = 1;
    tft.fillRect(BLACK);
    //Draw frame
    drawFrame(5, YELLOW);
    drawVerticalLine(125, RED);
    drawVerticalLine(195, GREEN);
    drawHorizontalLine(80, MAGENTA);
    drawHorizontalLine(150, BLUE);
    bPressed = false;
}
```

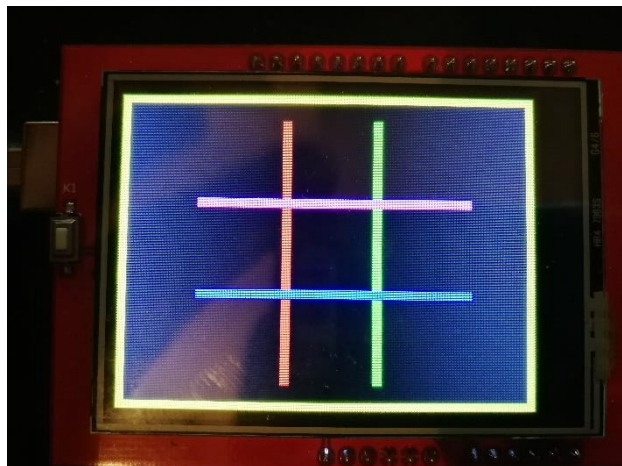


Abbildung 8: Buntes Spielfeld

Neben dem Generieren des Spielfelds wird auch die Funktion `playGame(bool)` aufgerufen. Diese Funktion bekommt über das Boolflag mitgegeben, ob der Spieler ein einfaches oder schweres Spiel spielen möchte und durchläuft die entsprechende Schleifenbedingung. Damit vom Quellcode nicht immer über die `loop`-Funktion in die Funktion `playGame` gesprungen werden muss, wird eine `Do-While`-Schleife genutzt. Diese hat zur Bedingung, dass noch kein Gewinner feststeht und nicht mehr als 9 Züge gemacht wurde, siehe Code 2.

```

//Start a loop to play game
void playGame(bool bHardMode)
{
    do
    {
        Serial.println("MOVE: " + String(iMoves)); //Print move
        if(!bHardMode) //Easy-Mode
        {
            if(iMoves % 2 == 1)
            {
                movePlayer();
                printBoard();
                checkWinner();
            }
            else
            {
                moveArduino(false);
                printBoard();
                checkWinner();
            }
        }
        else //Hard-Mode
        {
            if(iMoves % 2 == 1)
            {
                moveArduino(true);
                printBoard();
                checkWinner();
            }
            else
            {
                movePlayer();
                printBoard();
                checkWinner();
            }
        }
        iMoves++;
    }
    while(iWinner == 0 && iMoves < 10);
    //End of game, cause somebody win or no moves left
    if(iWinner == 1)
    {
        Serial.println("CPU WINS");
        delay(3000);
        drawGameEndScreen();
    }
    else if(iWinner == 2)
    {
        Serial.println("HUMAN WINS");
        delay(3000);
        drawGameEndScreen();
    }
    else
    {
        Serial.println("DRAW");
    }
}

```



Code 2: Ablauf von `playGame(bool)`

1. Spielzug Mensch / Arduino auf Durchführbarkeit prüfen
2. Spielzug auf dem Display darstellen
3. Ausgabe auf dem Seriellen Monitor der Spielfeldbelegung
4. Prüfen, ob es einen Gewinner gibt oder nicht

```

{
  bool bValidMove = false;
  Serial.println("Players move");
  do
  {
    bPressed = false;
    bPressed = Touch_getXY();
    if(bPressed)
    {
      Serial.println("X: " + String(pixel_x) + " Y: " + String(pixel_y));
      if((pixel_x<115)&& (pixel_y>=150)) //6
      {
        if(iSetMark[6]==0)
        {
          Serial.println("Player Move: 6");
          iSetMark[6] = 2;
          bValidMove = true;
          drawPlayerMove(6);
          Serial.println("Drawing player move");
        }
      }
      else if((pixel_x>0 && pixel_x<115)&& (pixel_y<150 && pixel_y>80)) //3
      {
        if(iSetMark[3]==0)
        {
          Serial.println("Player Move: 3");
          iSetMark[3] = 2;
          bValidMove = true;
          drawPlayerMove(3);
          Serial.println("Drawing player move");
        }
      }
      else if((pixel_x<125)&& (pixel_y<80)) //0
      {
        if(iSetMark[0]==0)
        {
          Serial.println("Player Move: 0");
        }
      }
    }
  } while(bPressed);
}

```

```

        iSetMark[0] = 2;
        bValidMove = true;
        drawPlayerMove(0);
    }
}
else if((pixel_x>125 && pixel_x<=195)&& (pixel_y<80)) //1
{
    if(iSetMark[1]==0)
    {
        Serial.println("Player Move: 1");
        iSetMark[1] = 2;
        bValidMove = true;
        drawPlayerMove(1);
    }
}
else if((pixel_x>195)&& (pixel_y<80)) //2
{
    if(iSetMark[2]==0)
    {
        Serial.println("Player Move: 2");
        iSetMark[2] = 2;
        bValidMove = true;
        drawPlayerMove(2);
    }
}
else if((pixel_x>125 && pixel_x<=195)&& (pixel_y<150 && pixel_y>80)) //4
{
    if(iSetMark[4]==0)
    {
        Serial.println("Player Move: 4");
        iSetMark[4] = 2;
        bValidMove = true;
        drawPlayerMove(4);
    }
}
else if((pixel_x>195)&& (pixel_y<150 && pixel_y>80)) //5
{
    if(iSetMark[5]==0)
    {
        Serial.println("Player Move: 5");
        iSetMark[5] = 2;
        bValidMove = true;
        drawPlayerMove(5);
    }
}
else if((pixel_x>125 && pixel_x<=195)&& (pixel_y>150)) //7
{
    if(iSetMark[7]==0)
    {
        Serial.println("Player Move: 7");
        iSetMark[7] = 2;
        bValidMove = true;
        drawPlayerMove(7);
    }
}
else if((pixel_x>195)&& (pixel_y>150)) //8

```

```

    {
        if(iSetMark[8]==0)
        {
            Serial.println("Player Move: 8");
            iSetMark[8] = 2;
            bValidMove = true;
            drawPlayerMove(8);
        }
    }
}
}while(!bValidMove);
}

```

*Code 3: Funktion movePlayer()*

Bei der Funktion moveArduino(bool) übergibt das Boolflag der Funktion den Spielmodus, einfaches oder schweres Spiel. Beim einfachen Spiel wird ein zufälliger Zahlenwert von 0 bis 8 generiert und geprüft, ob dieses Spielfeld schon einmal gespielt wurde. Sollte das der Fall sein, so wird die Do-While-Schleife solange durchlaufen, bis eine Zufallszahl generiert wurde, die auf dem Spielfeld noch nicht gewählt wurde, siehe Code 4.

Bei der Spieloption schwer passieren gleich mehrere Dinge. Zunächst gehört der erste Spielzug dem Arduino, der mittels eines Modulo entscheidet, ob direkt die Position 4, also Mitte oder ein zufälliges Feld vom Arduino gewählt wird, siehe Code 4 im Quellcode. Bei allen weiteren Zügen wird immer geprüft, ob der Spieler am Gewinnen gehindert werden muss oder aber der Arduino selbst gewinnen kann.

```

//Function to let Arduino make a move
void moveArduino(bool bHardMode)
{
    bool bValidMove = false;
    int iRandomMove;
    Serial.println("Arduino move");
    //Hard mode
    if(bHardMode)
    {
        //First move Arduino draw direct to middle or sometimes other pos
        if(iMoves == 1)
        {
            if(millis() % 8 == 0) //Thats why other position is possible
            {
                iRandomMove = random(9);
                iSetMark[iRandomMove] = 1;
                Serial.println("Arduino Move: " + String(iRandomMove));
                drawArduinoMove(iRandomMove);
            }
            else
            {
                iSetMark[4] = 1;
                Serial.println("Arduino Move: " + String(4));
                drawArduinoMove(4);
            }
        }
        else
        {
            int iNextMove = checkPlayerMove();
            int iWinMove = checkPossibleWin();

```

```

Serial.println("Check player: " + String(iWinMove));
if(iWinMove >= 0)
{
    delay(1000);
    iSetMark[iWinMove] = 1;
    Serial.println("Arduino Move: " + String(iWinMove));
    drawArduinoMove(iWinMove);
}
else
{
    if(iNextMove >= 0)
    {
        iSetMark[iNextMove] = 1;
        Serial.println("Arduino Move: " + String(iNextMove));
        drawArduinoMove(iNextMove);
    }
    else
    {
        do{
            iRandomMove = random(9);
            if(iSetMark[iRandomMove] == 0)
            {
                delay(1000);
                iSetMark[iRandomMove] = 1;
                Serial.println("Arduino Move: " + String(iRandomMove));
                drawArduinoMove(iRandomMove);
                bValidMove = true;
            }
        }while(!bValidMove);
    }
}
} //else
} //if(bHardMode)
else //Easy Mode
{
    do{
        iRandomMove = random(9);
        if(iSetMark[iRandomMove] == 0)
        {
            delay(1000);
            iSetMark[iRandomMove] = 1;
            Serial.println("Arduino Move: " + String(iRandomMove));
            drawArduinoMove(iRandomMove);
            bValidMove = true;
        }
    }while(!bValidMove);
}
}
}

```

Code 4: Funktion moveArduino (bool)

In beiden Fällen wird eine Zeichenfunktion, drawPlayerMove(int) bzw. drawArduinoMove(int), aufgerufen. Der Integer ist dabei die Spielfeldposition, die in eine X- und Y-Koordinate auf dem Display übersetzt werden muss. Die jeweilige Funktion ruft wiederum die Zeichenfunktion drawSign() auf, welche das X- bzw. O-Zeichen an die entsprechende Stelle im Spielfeld zeichnet.

Sie werden sich nun fragen, was das `x_bitmap` bzw. `circle` bei den beiden vorherig genannten Funktionen ist. Hier kommt die Datei `graphics.c` zum Einsatz, in der für das X- bzw. O-Zeichen ein Array hinterlegt ist. Diese beiden Arrays werden in der Zeile 23 und 24 in der Variable `circle` und `x_bitmap` geladen, siehe Abbildung 9.

```
extern uint8_t circle[];  
extern uint8_t x_bitmap[];
```

Abbildung 9: Array für die Zeichen laden

Das Schlagwort `extern`, verweist dabei auf eine externe C-Datei, die sich im gleichen Projektordner wie die `*.ino`-Datei befinden muss. Danach wird das Zeichen in der Funktion `drawSign()` Bit für Bit ausgelesen und mit der angegebenen Farbe an der richtigen X- und Y-Koordinate vom Touchdisplay eingefügt.

Wollen Sie auch hier individuell sein, so können Sie in der Funktion `drawPlayerMove(int)` bzw. `drawArduinoMove(int)`, siehe Zeile 563 bis 596 im Quellcode, entsprechend die Farbe vom Zeichen verändern.

Damit ist der größte Teil vom Programm erklärt.

Punkt 3 ist eine simple Ausgabe mittels For-Schleife auf dem Seriellen Monitor. Punkt 4 der obigen Liste, wird über die Funktion `checkWinner()` realisiert, die jede Kombination für eine Gewinnerreihe prüft, ob Sie oder der Arduino eine Reihe hat und entsprechend die globale Variable `iWinner` überschreibt.

Sobald `iWinner` überschrieben oder 9 Züge erreicht wurden, springt das Programm aus der Do-While-Schleife von `playGame(bool)` und verkündet mit der Funktion `drawGameEndScreen()` den Gewinner oder das Unentschieden (engl. draw), siehe Abbildung 10. Damit Sie vorher noch sehen, warum Sie oder der Arduino gewonnen haben, wird eine Verzögerung von 3 Sekunden angesetzt. Gleiches gilt auch bei einem Unentschieden.



Abbildung 10: Endscreen mit Gewinner

Mit dem GameEnd-Screen werden auch wieder die beiden Buttons zur Auswahl von einem leichten oder schweren Spiel eingeblendet.

Gerne können Sie auch nach Lust und Laune den Quellcode modifizieren. Denkbar wäre, dass der Schwierigkeitsgrad schwer ein bisschen einfacher wird, indem ab und zu der Arduino eher einen zufälligen Zug macht, anstatt immer zu versuchen Sie zu blocken. Hierzu müssten Sie ggf. wieder mit einem Modulo arbeiten.

Sie können aber auch einfach an den aufgezeigten Stellen die Farbe der Elemente verändern, um so einen individuellen Touch zu erhalten.

Ich wünsche Ihnen viel Spaß beim Nachbau.



Dieses und weitere Projekte finden sich auf GitHub unter  
<https://github.com/M3taKn1ght/Blog-Repo>