Tic Tac Toe & Reaktionsspiel

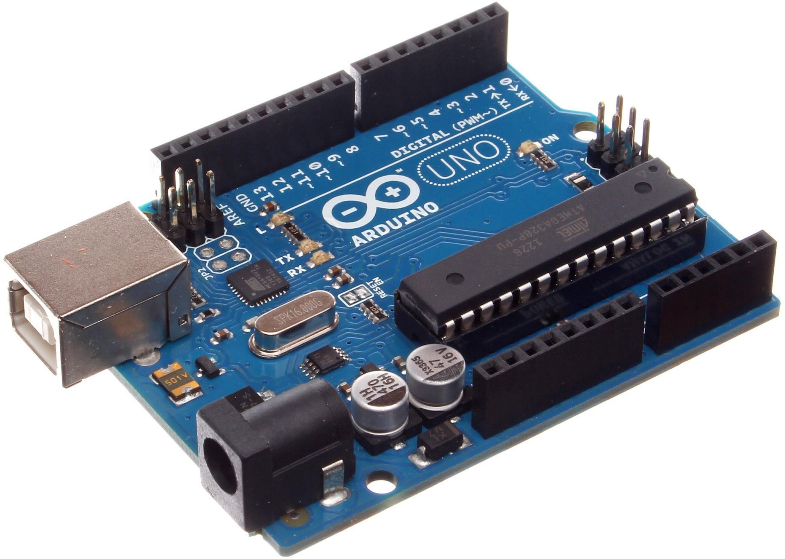
Uli Stein, Clemens Hanselmann, Nico Braun & Daniel Huber

Inhalt

- Aufgabenstellung
- Komponenten
- Hardwareaufbau
- Ansteuerung der Fernbedienung
- Ansteuerung der Matrix
- Ansteuerung der 7 Segmentanzeige
- Spiellogik Tic Tac Toe
- Spiellogik Reaktionsspiel

Komponenten

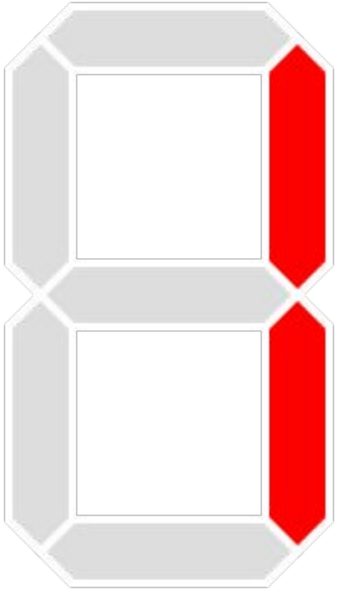
Arduino



Komponenten

7 Segment

- 7-Segment-Anzeige
- 7 Anschlüsse für jedes Segment einen

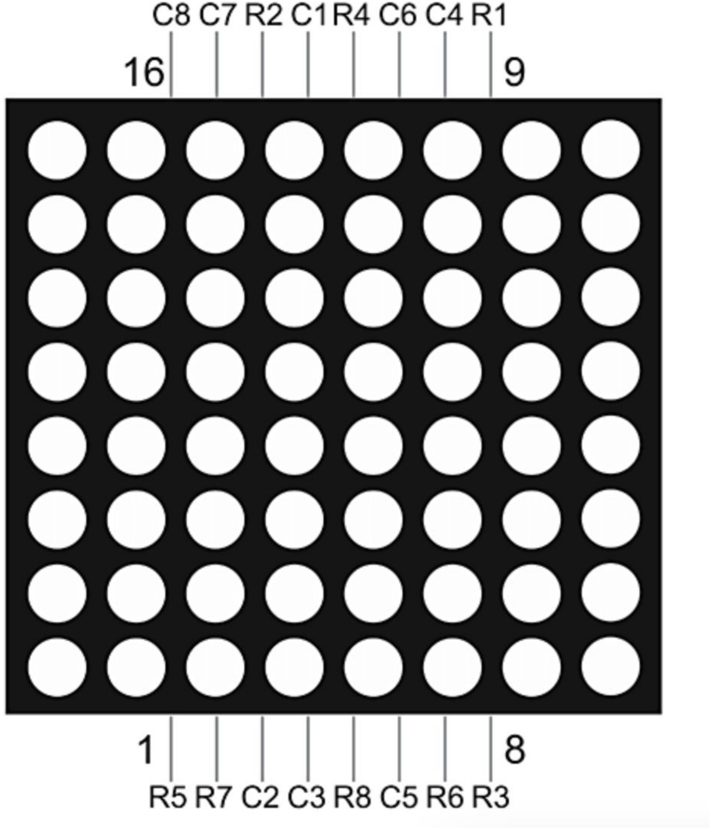


Komponenten

8*8 Matrix

Ansteuerung über Reihen und Spalten

Multiplexing für Muster



Komponenten

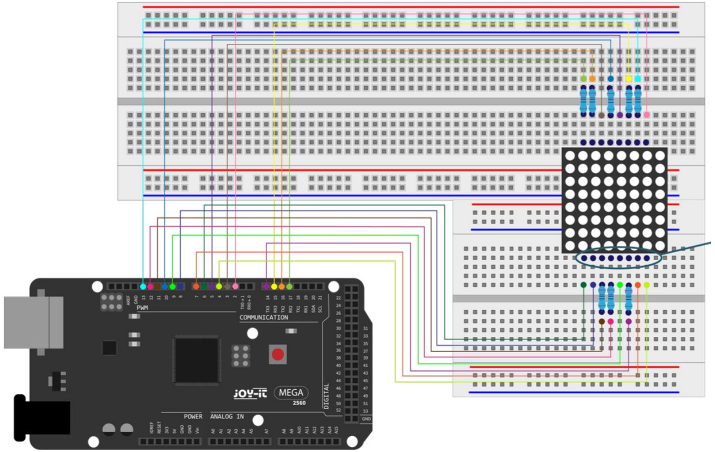
Fernbedienung

- Ir - Fernbedienung
- sendet Infrarotsignal
- Signal muss für Spiel in Dezimal

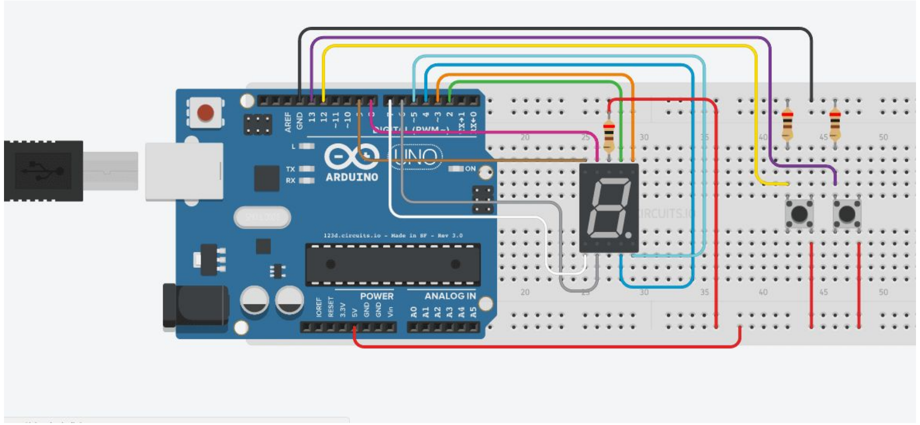
  umgewandelt werden

Hardware Aufbau

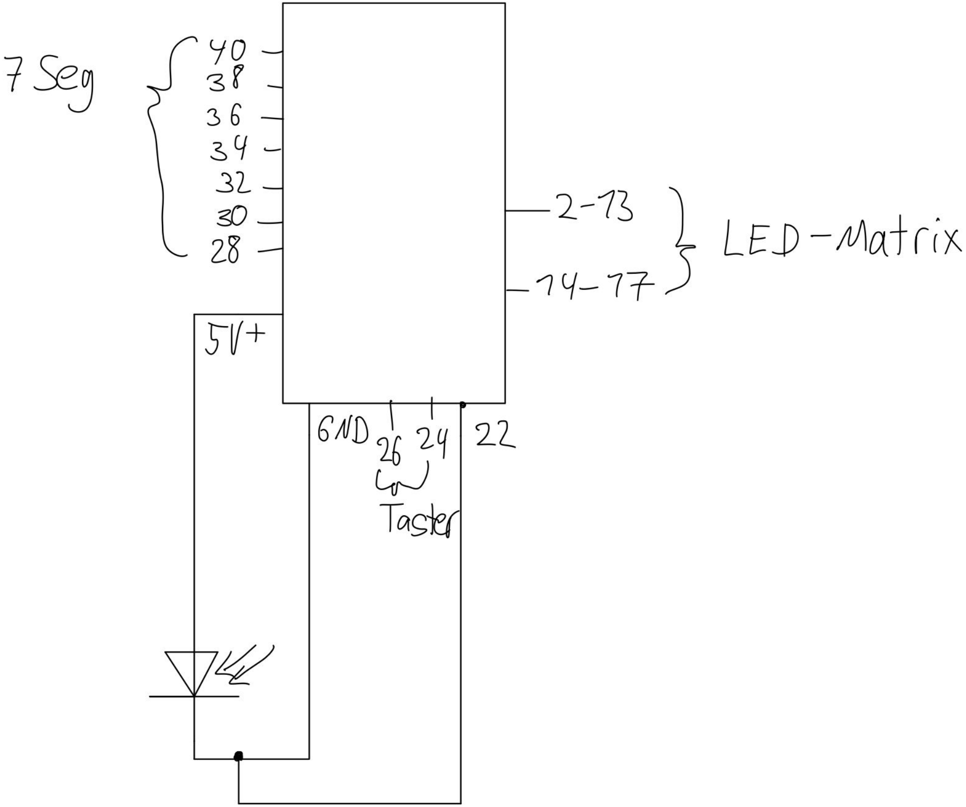
Matrix



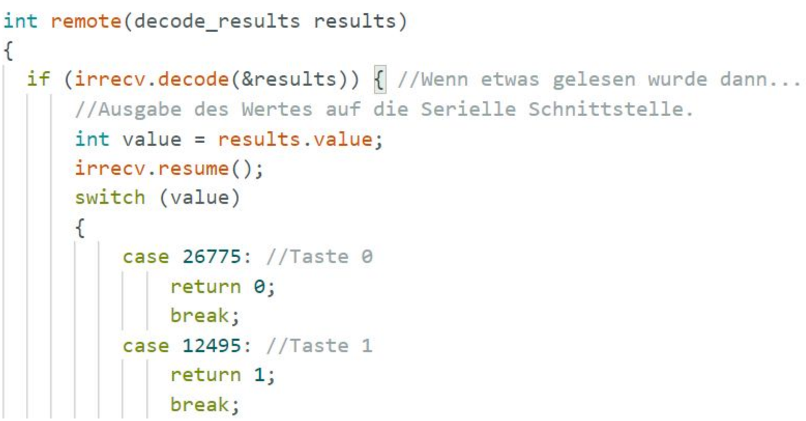
7 Segment



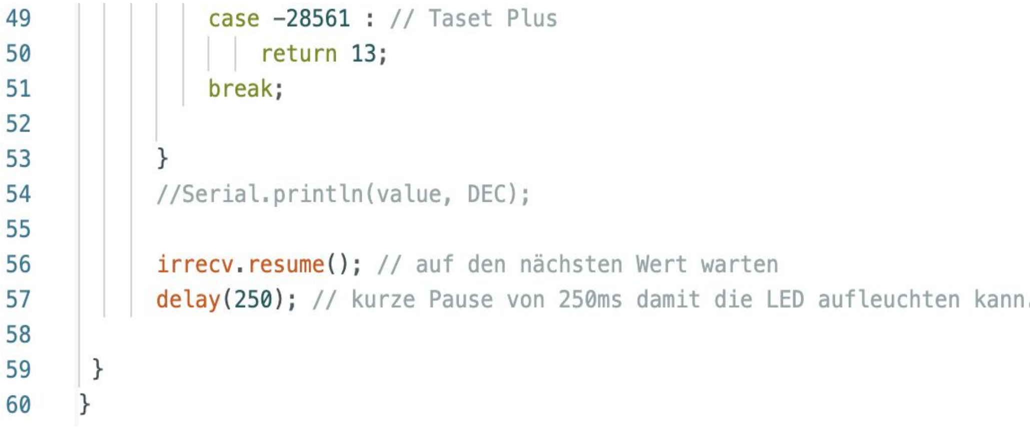
Schaltplan


7 Seg
40
38
36
34
32
30
28
2-13
14-17
LED-Matrix
5V+
GND
26
24
22
Taster

Ansteuerung der Fernbedienung

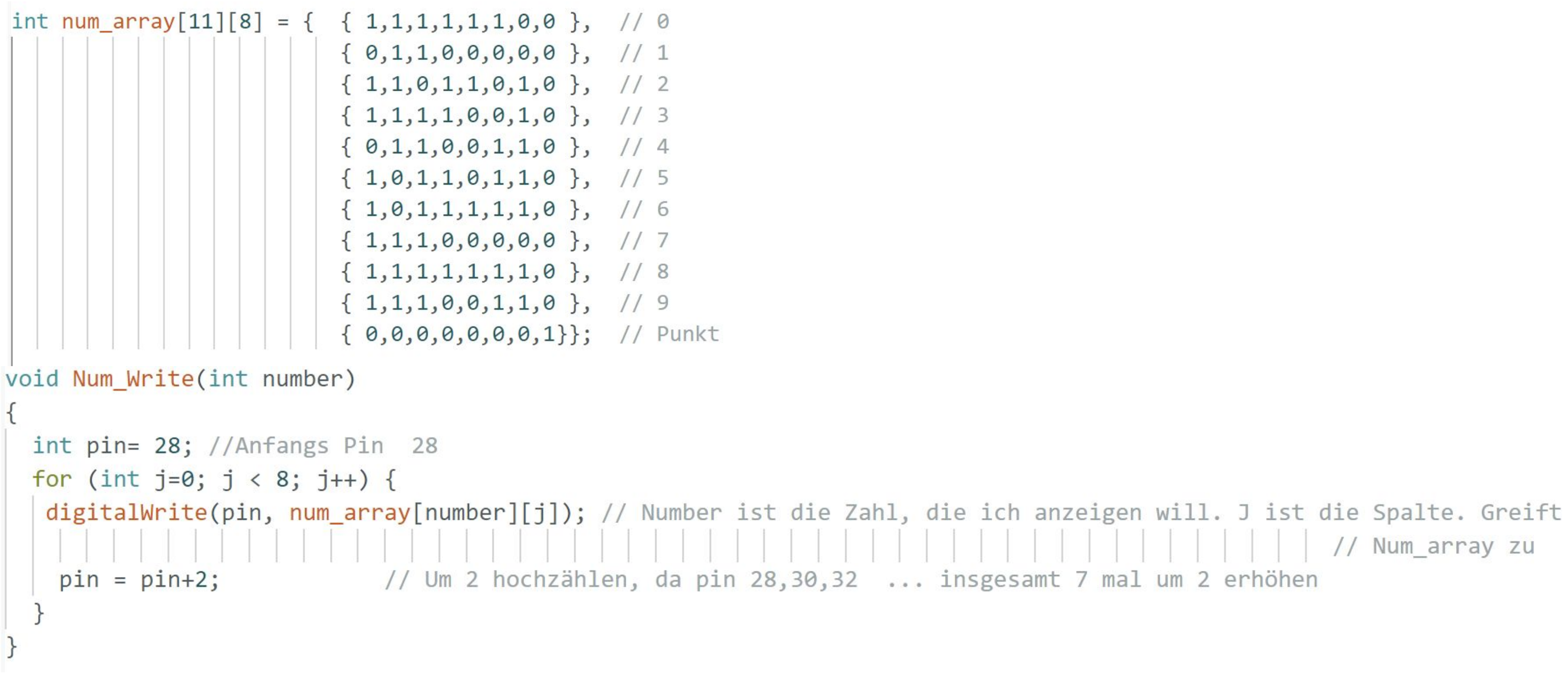


```
int remote(decode_results results)
{
  if (irrecv.decode(&results)) { //Wenn etwas gelesen wurde dann...
      //Ausgabe des Wertes auf die Serielle Schnittstelle.
      int value = results.value;
      irrecv.resume();
      switch (value)
      {
          case 26775: //Taste 0
              return 0;
              break;
          case 12495: //Taste 1
              return 1;
              break;
```



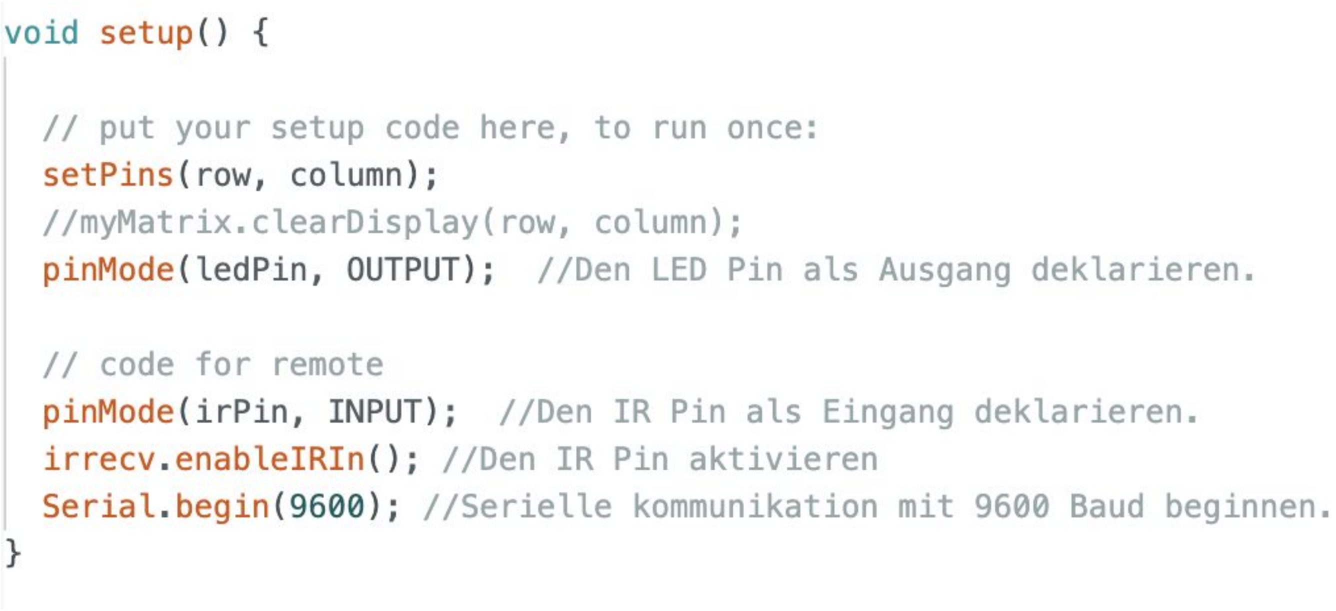
```
            case -28561 : // Taset Plus
                return 13;
            break;

        }
        //Serial.println(value, DEC);

        irrecv.resume(); // auf den nächsten Wert warten
        delay(250); // kurze Pause von 250ms damit die LED aufleuchten kann

  }
}
```

Ansteuerung der 7 Segmentanzeige



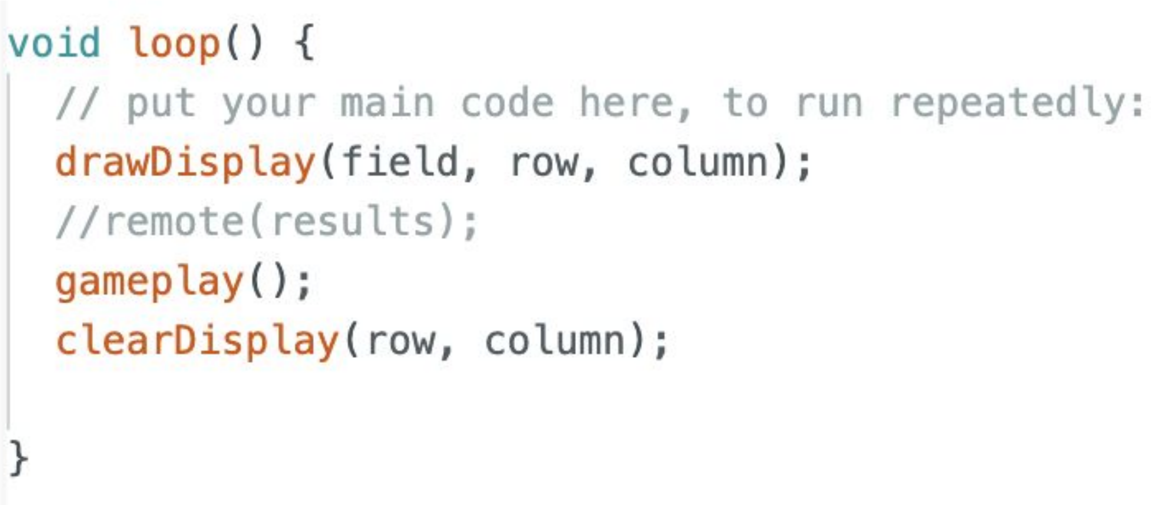
```
int num_array[11][8] = {  { 1,1,1,1,1,1,0,0 },  // 0
                          { 0,1,1,0,0,0,0,0 },  // 1
                          { 1,1,0,1,1,0,1,0 },  // 2
                          { 1,1,1,1,0,0,1,0 },  // 3
                          { 0,1,1,0,0,1,1,0 },  // 4
                          { 1,0,1,1,0,1,1,0 },  // 5
                          { 1,0,1,1,1,1,1,0 },  // 6
                          { 1,1,1,0,0,0,0,0 },  // 7
                          { 1,1,1,1,1,1,1,0 },  // 8
                          { 1,1,1,0,0,1,1,0 },  // 9
                          { 0,0,0,0,0,0,0,1}};  // Punkt
void Num_Write(int number)
{
  int pin= 28; //Anfangs Pin  28
  for (int j=0; j < 8; j++) {
   digitalWrite(pin, num_array[number][j]); // Number ist die Zahl, die ich anzeigen will. J ist die Spalte. Greift
                                                                                                    // Num_array zu
    pin = pin+2;            // Um 2 hochzählen, da pin 28,30,32  ... insgesamt 7 mal um 2 erhöhen
  }
}
```

Setup



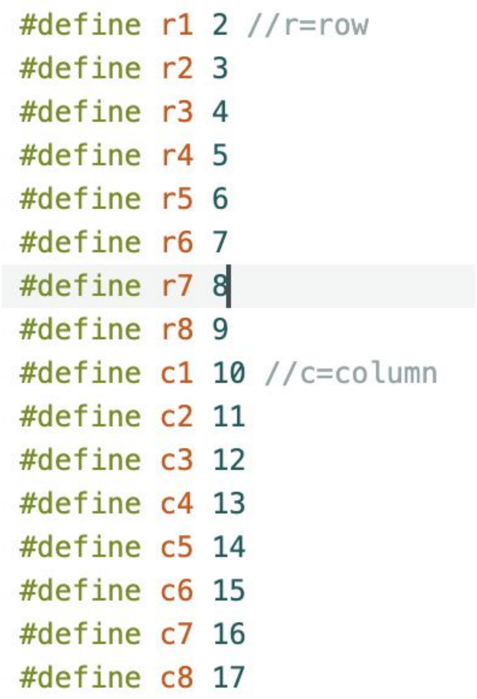
```
void setup() {

  // put your setup code here, to run once:
  setPins(row, column);
  //myMatrix.clearDisplay(row, column);
  pinMode(ledPin, OUTPUT);  //Den LED Pin als Ausgang deklarieren.

  // code for remote
  pinMode(irPin, INPUT);  //Den IR Pin als Eingang deklarieren.
  irrecv.enableIRIn(); //Den IR Pin aktivieren
  Serial.begin(9600); //Serielle kommunikation mit 9600 Baud beginnen.
}
```

Loop

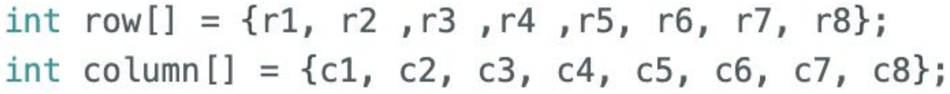


```
void loop() {
  // put your main code here, to run repeatedly:
  drawDisplay(field, row, column);
  //remote(results);
  gameplay();
  clearDisplay(row, column);

}
```

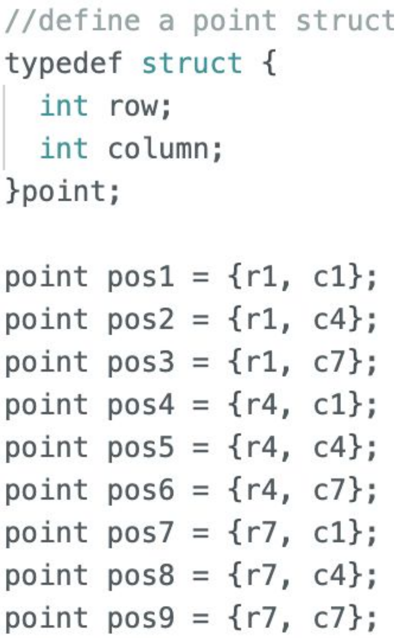
Ansteuerung der Matrix



```
#define r1 2 //r=row
#define r2 3
#define r3 4
#define r4 5
#define r5 6
#define r6 7
#define r7 8
#define r8 9
#define c1 10 //c=column
#define c2 11
#define c3 12
#define c4 13
#define c5 14
#define c6 15
#define c7 16
#define c8 17
```

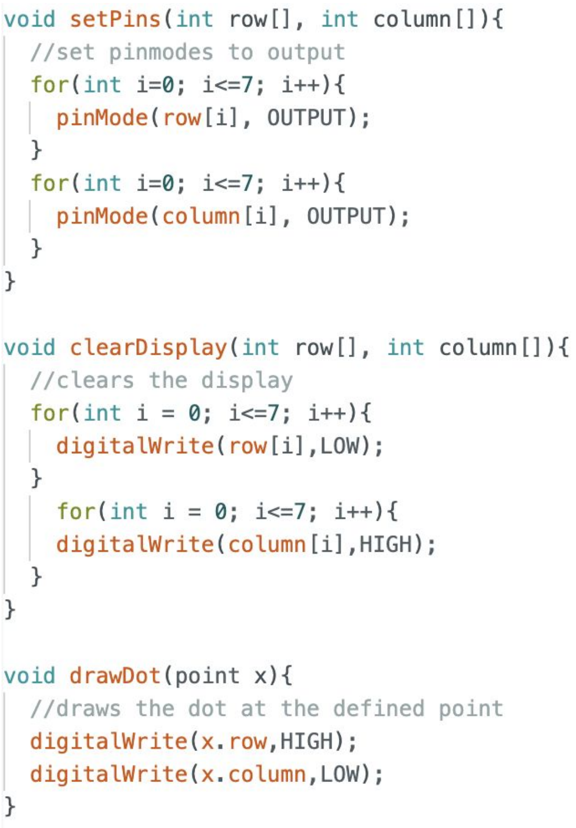


```
int row[] = {r1, r2 ,r3 ,r4 ,r5, r6, r7, r8};
int column[] = {c1, c2, c3, c4, c5, c6, c7, c8};
```

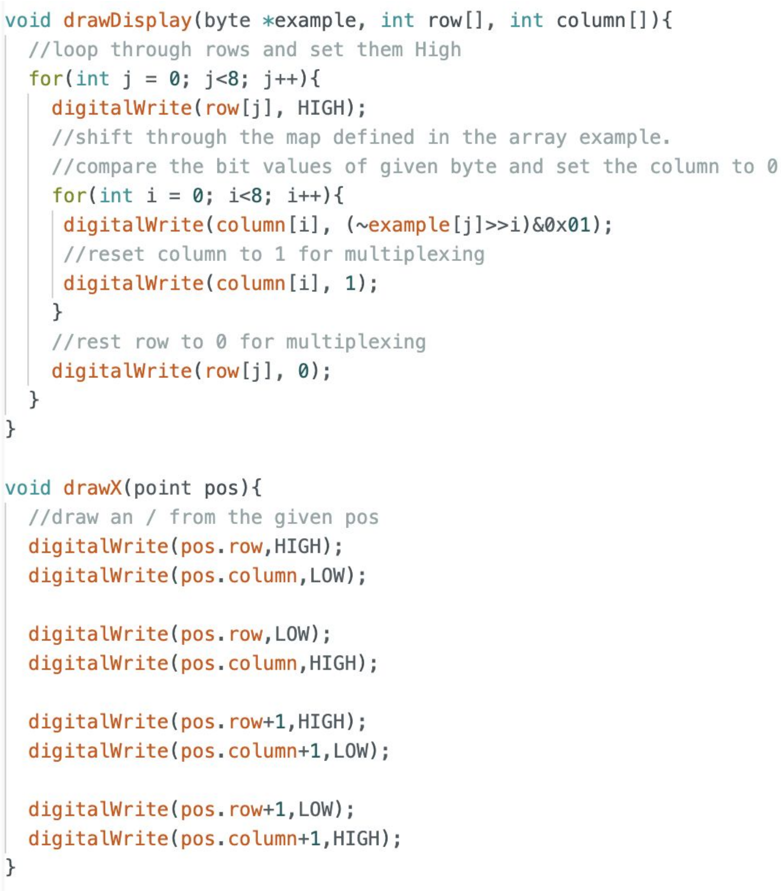


```
//define a point struct
typedef struct {
  int row;
  int column;
}point;

point pos1 = {r1, c1};
point pos2 = {r1, c4};
point pos3 = {r1, c7};
point pos4 = {r4, c1};
point pos5 = {r4, c4};
point pos6 = {r4, c7};
point pos7 = {r7, c1};
point pos8 = {r7, c4};
point pos9 = {r7, c7};
```

Ansteuerung der Matrix

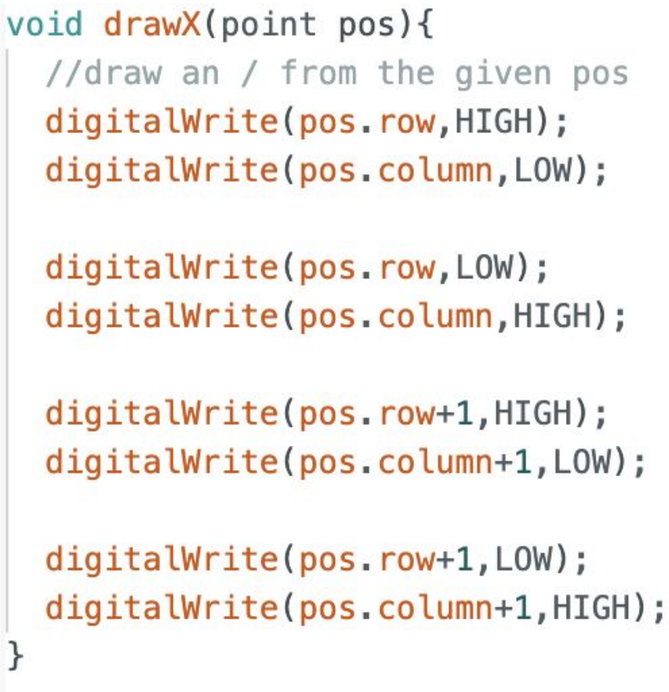


```
void setPins(int row[], int column[]){
  //set pinmodes to output
  for(int i=0; i<=7; i++){
    pinMode(row[i], OUTPUT);
  }
  for(int i=0; i<=7; i++){
    pinMode(column[i], OUTPUT);
  }
}

void clearDisplay(int row[], int column[]){
  //clears the display
  for(int i = 0; i<=7; i++){
    digitalWrite(row[i],LOW);
  }
    for(int i = 0; i<=7; i++){
    digitalWrite(column[i],HIGH);
  }
}

void drawDot(point x){
  //draws the dot at the defined point
  digitalWrite(x.row,HIGH);
  digitalWrite(x.column,LOW);
}
```



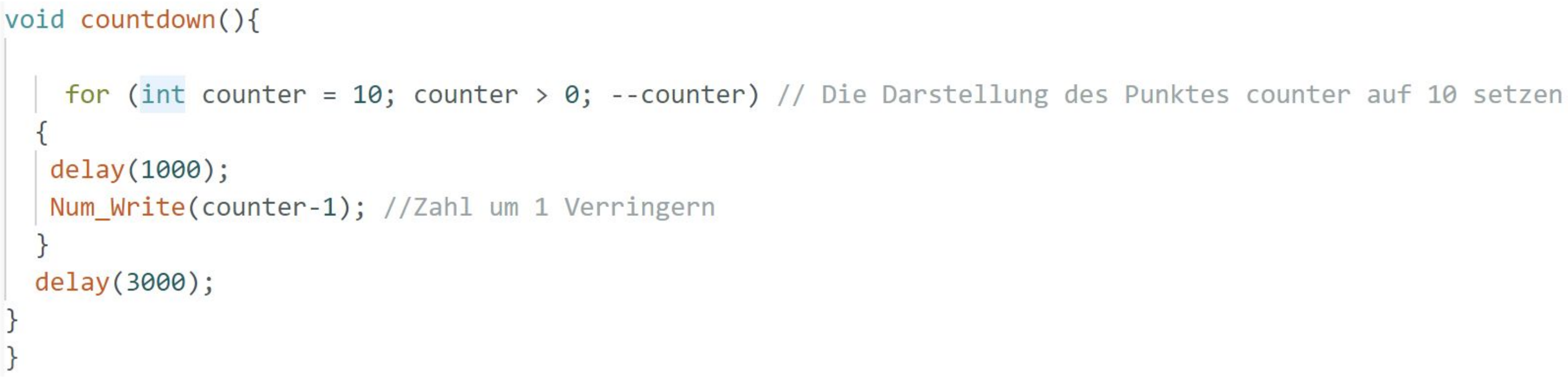
```
void drawDisplay(byte *example, int row[], int column[]){
  //loop through rows and set them High
  for(int j = 0; j<8; j++){
    digitalWrite(row[j], HIGH);
    //shift through the map defined in the array example.
    //compare the bit values of given byte and set the column to 0
    for(int i = 0; i<8; i++){
     digitalWrite(column[i], (~example[j]>>i)&0x01);
     //reset column to 1 for multiplexing
     digitalWrite(column[i], 1);
    }
    //rest row to 0 for multiplexing
    digitalWrite(row[j], 0);
  }
}

void drawX(point pos){
  //draw an / from the given pos
  digitalWrite(pos.row,HIGH);
  digitalWrite(pos.column,LOW);

  digitalWrite(pos.row,LOW);
  digitalWrite(pos.column,HIGH);

  digitalWrite(pos.row+1,HIGH);
  digitalWrite(pos.column+1,LOW);

  digitalWrite(pos.row+1,LOW);
  digitalWrite(pos.column+1,HIGH);
}
```

Ansteuerung der Matrix



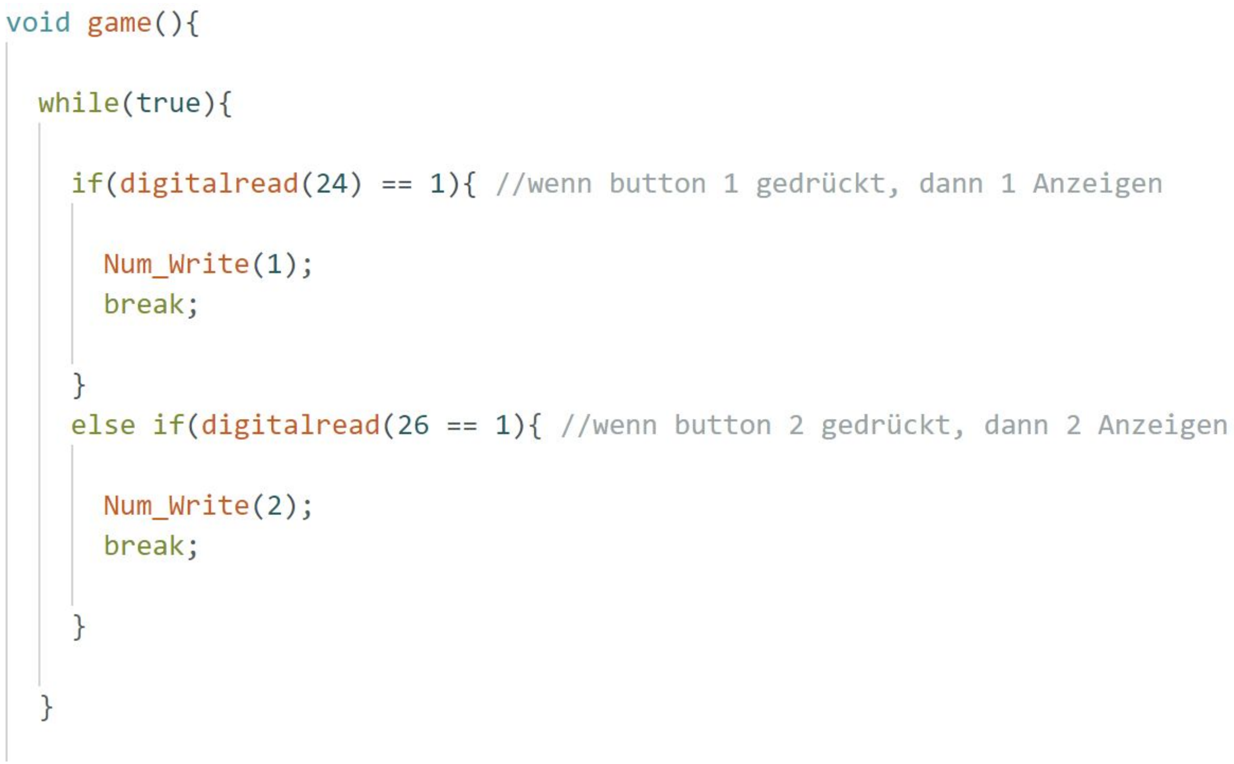
```
void drawX(point pos){
  //draw an / from the given pos
  digitalWrite(pos.row,HIGH);
  digitalWrite(pos.column,LOW);

  digitalWrite(pos.row,LOW);
  digitalWrite(pos.column,HIGH);

  digitalWrite(pos.row+1,HIGH);
  digitalWrite(pos.column+1,LOW);

  digitalWrite(pos.row+1,LOW);
  digitalWrite(pos.column+1,HIGH);
}
```

Spiellogik Reaktionsspiel



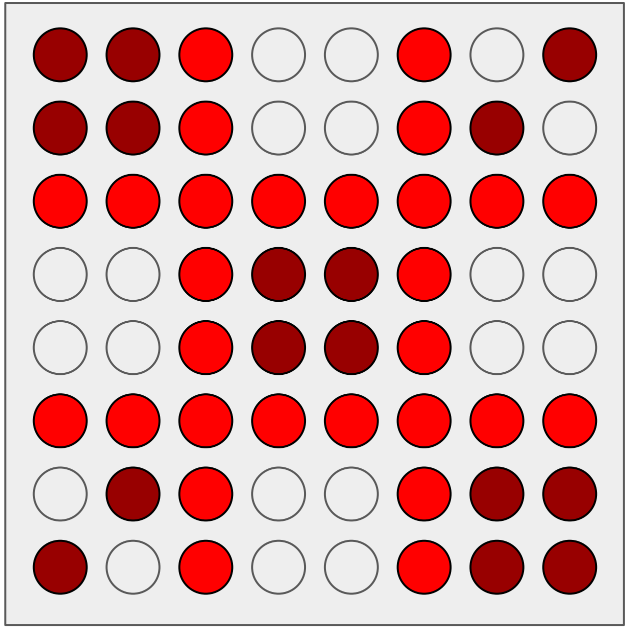
```
void countdown(){

    for (int counter = 10; counter > 0; --counter) // Die Darstellung des Punktes counter auf 10 setzen
  {
   delay(1000);
   Num_Write(counter-1); //Zahl um 1 Verringern
  }
  delay(3000);
}
}
```

Spiellogik Reaktionsspiel

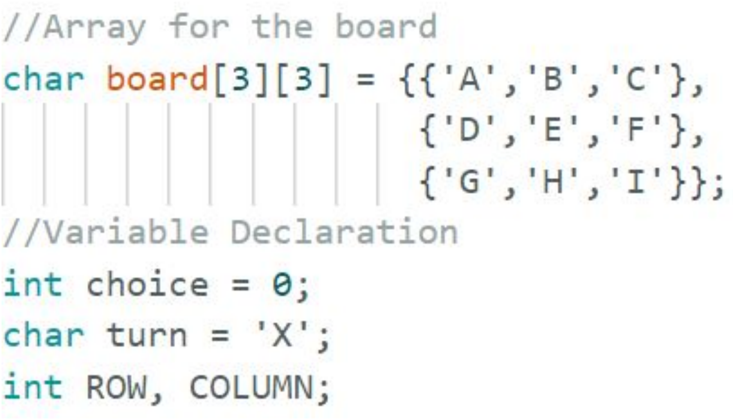


```
void game(){

  while(true){

    if(digitalread(24) == 1){ //wenn button 1 gedrückt, dann 1 Anzeigen

      Num_Write(1);
      break;

    }
    else if(digitalread(26 == 1){ //wenn button 2 gedrückt, dann 2 Anzeigen

      Num_Write(2);
      break;

    }

  }
```

Spielfeld TickTackToe

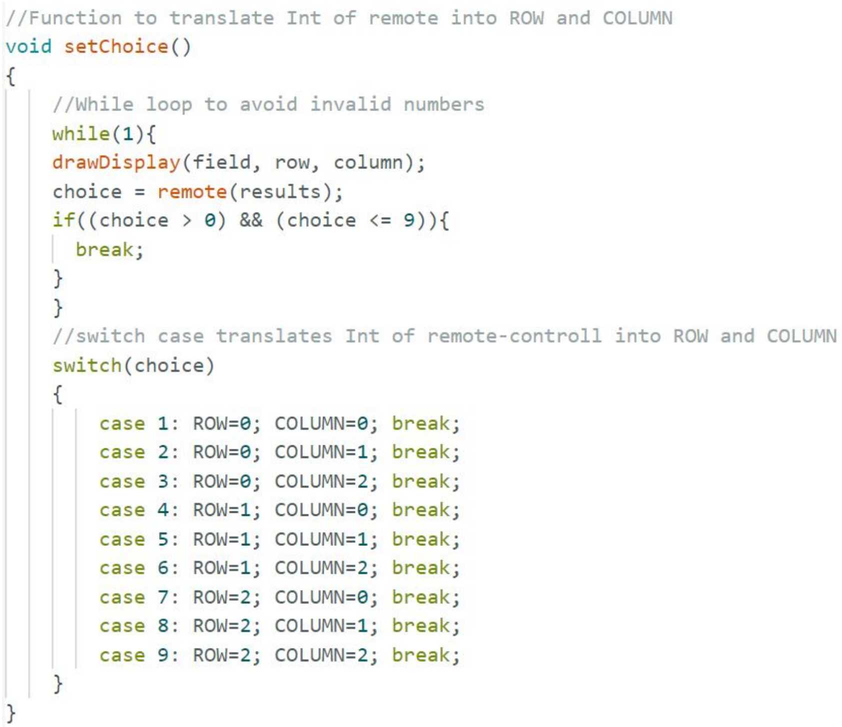


Spiellogik TicTacToe

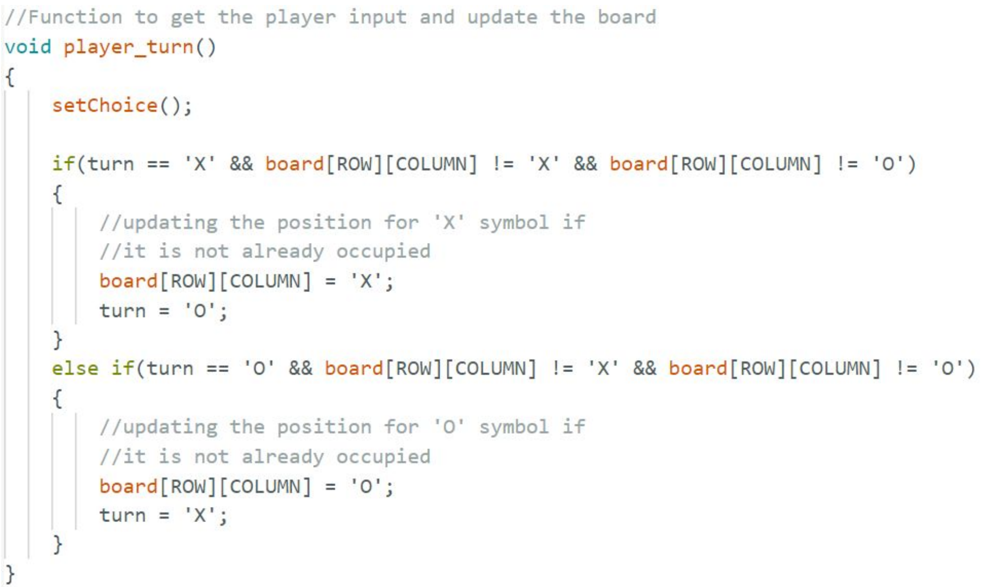


```
//Array for the board
char board[3][3] = {{'A','B','C'},
                    {'D','E','F'},
                    {'G','H','I'}};
//Variable Declaration
int choice = 0;
char turn = 'X';
int ROW, COLUMN;
```

Spiellogik TicTacToe

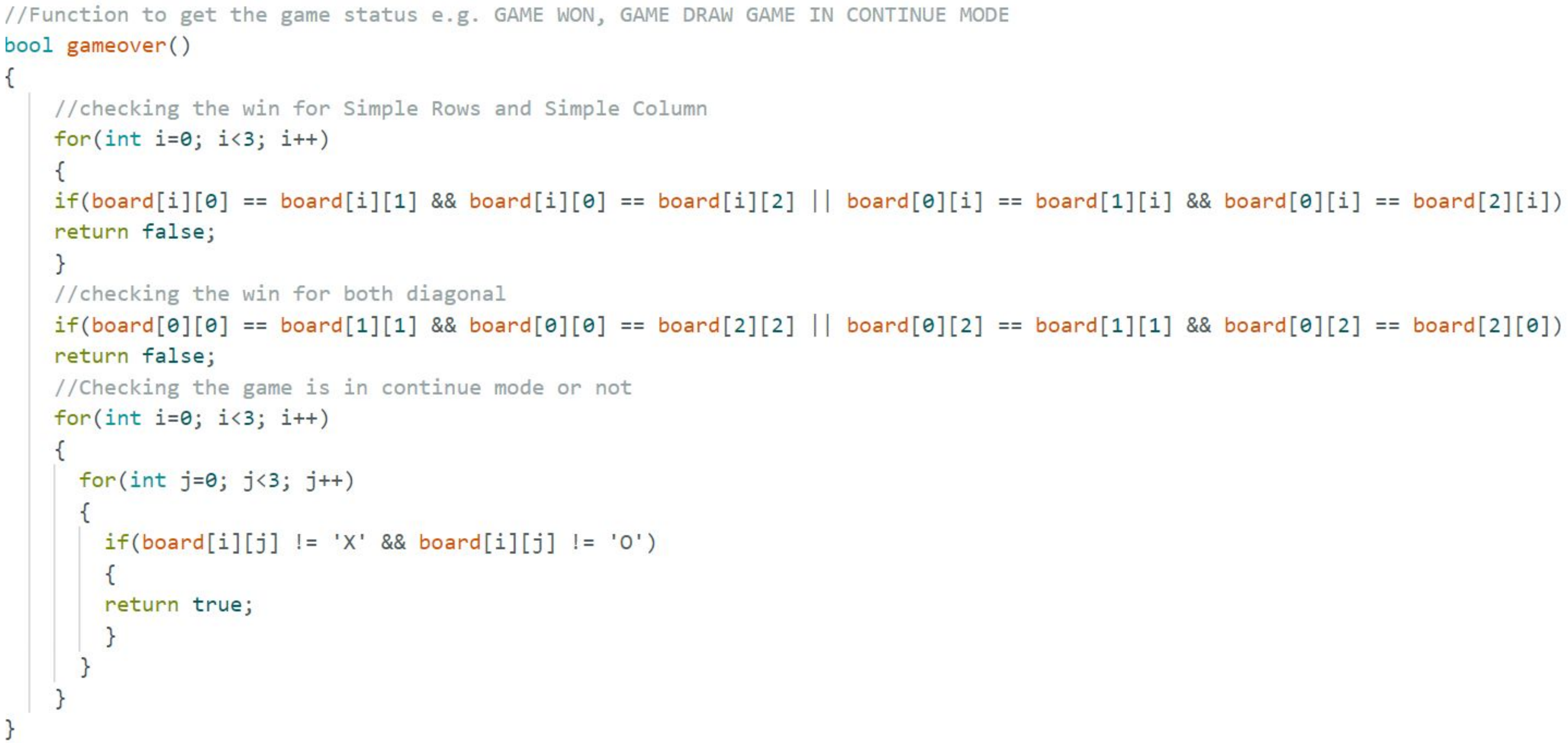


```
//Function to translate Int of remote into ROW and COLUMN
void setChoice()
{
    //While loop to avoid invalid numbers
    while(1){
    drawDisplay(field, row, column);
    choice = remote(results);
    if((choice > 0) && (choice <= 9)){
      break;
    }
    }
    //switch case translates Int of remote-controll into ROW and COLUMN
    switch(choice)
    {
        case 1: ROW=0; COLUMN=0; break;
        case 2: ROW=0; COLUMN=1; break;
        case 3: ROW=0; COLUMN=2; break;
        case 4: ROW=1; COLUMN=0; break;
        case 5: ROW=1; COLUMN=1; break;
        case 6: ROW=1; COLUMN=2; break;
        case 7: ROW=2; COLUMN=0; break;
        case 8: ROW=2; COLUMN=1; break;
        case 9: ROW=2; COLUMN=2; break;
    }
}
```



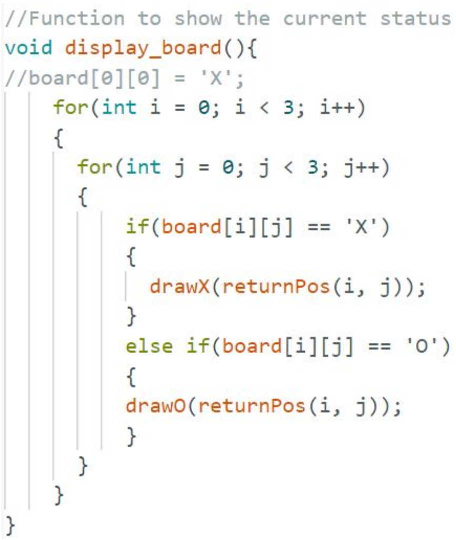
```
//Function to get the player input and update the board
void player_turn()
{
    setChoice();

    if(turn == 'X' && board[ROW][COLUMN] != 'X' && board[ROW][COLUMN] != 'O')
    {
        //updating the position for 'X' symbol if
        //it is not already occupied
        board[ROW][COLUMN] = 'X';
        turn = 'O';
    }
    else if(turn == 'O' && board[ROW][COLUMN] != 'X' && board[ROW][COLUMN] != 'O')
    {
        //updating the position for 'O' symbol if
        //it is not already occupied
        board[ROW][COLUMN] = 'O';
        turn = 'X';
    }
}
```

Spiellogik TicTacToe

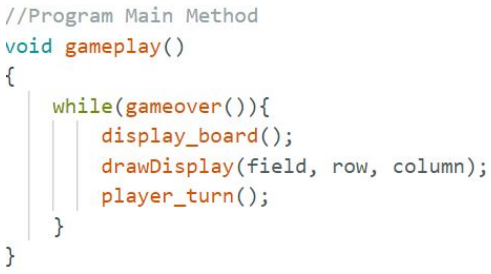


```
//Function to get the game status e.g. GAME WON, GAME DRAW GAME IN CONTINUE MODE
bool gameover()
{
    //checking the win for Simple Rows and Simple Column
    for(int i=0; i<3; i++)
    {
    if(board[i][0] == board[i][1] && board[i][0] == board[i][2] || board[0][i] == board[1][i] && board[0][i] == board[2][i])
    return false;
    }
    //checking the win for both diagonal
    if(board[0][0] == board[1][1] && board[0][0] == board[2][2] || board[0][2] == board[1][1] && board[0][2] == board[2][0])
    return false;
    //Checking the game is in continue mode or not
    for(int i=0; i<3; i++)
    {
      for(int j=0; j<3; j++)
      {
        if(board[i][j] != 'X' && board[i][j] != 'O')
        {
        return true;
        }
      }
    }
}
```

Spiellogik TicTacToe



```
//Function to show the current status
void display_board(){
//board[0][0] = 'X';
  for(int i = 0; i < 3; i++)
  {
   for(int j = 0; j < 3; j++)
   {
     if(board[i][j] == 'X')
     {
      drawX(returnPos(i, j));
     }
     else if(board[i][j] == 'O')
     {
     drawO(returnPos(i, j));
     }
   }
  }
}
```



```
//Program Main Method
void gameplay()
{
  while(gameover()){
    display_board();
    drawDisplay(field, row, column);
    player_turn();
  }
}
```