Beleg im Modul Embedded Systems II

Bau einer LED-Spielematte

Rauch, Tobias

Matrikel-Nr.: 78309

Prof. Dr.-Ing. Andreas Pretschner

Abgabe: 31.03.2024

1 Idee

Die LED-Spielmatte wurde ursprünglich für die Lange Nacht des Coding entwickelt. Die Idee, eine LED-Matte aus einzelnen LED-Paneelen mit einzeln adressierbaren RGB-LEDs. Auf dieser Matte sollen Spiele wie Pong und Snake gespielt werden können. Die Steuerung der Schlange bzw. der Schläger, soll über Balanceboards erfolgen. Die Kommunikation zwischen den Balanceboards und der Spielekonsole soll über WLAN funktionieren. Genauer gesagt soll die Kommunikation per UDP erfolgen.

2 Umsetzung

2.1 LED-Matte

Die LED-Matte wurde aus 25 einzelnen LED-Paneelen zusammengesetzt. Jedes LED-Paneel besitzt 256 einzeln adressierbare LEDs, welche in einem Rastermaß von 16 x 16 LEDs angeordnet ist. Insgesamt besitzt die LED-Matte 6.400 LEDs.

Die verbauten LEDs auf den vorgefertigten LED-Matten sind sogenannte Neopixel-LEDs. Neopixel-LEDs haben den Vorteil, dass jede einzelne LED separat durch einen Mikrocontroller über eine einzige Datenleitung angesteuert werden kann. Somit können Farbe und Helligkeit von jeder LED einzeln programmiert und ausgegeben werden können. Diese LEDs sind mit unterschiedlichen Treibermodulen für unterschiedliche Spannungen ausgelegt. Bei dieser Umsetzung handelt es sich um die WS2812B-Neopixel RGB LEDs, welche eine 5 V Spannung benötigen. Eine solche LED besteht hierbei aus vier Beinen, dem Plus und Minus-Pol der Spannungsversorgung, sowie einem Dateneingang und einem Datenausgang.

Laut Datenblatt des Herstellers benötigt eine LED, bei voller Helligkeit und in der Farbe Weiß 60 mA Strom. Dies würde bedeuten, wenn alle 6.400 LEDs in voller Helligkeit und in Weiß leuchten, eine Gesamtleistung von 1.920 W notwendig wäre.

Um alle 6.400 LEDs mit genügend Spannung zu versorgen wurden 5 Netzteile, mit 200 W verbaut. Somit steht eine Gesamtleistung von maximal 1.000 Watt zur Verfügung. Dies setzt voraus das alle LEDs niemals in voller Helligkeit leuchten dürfen, da sonst die Leistung der Netzteile für die LED-Matten nicht ausreichen würde.

Jedes Netzteil versorgt jeweils fünf LED-Matten. Dies entspricht 1.280 LEDs. Hierbei wurden die LED-Matten parallel geschalten. Für die LED-Paneele wurden Kabel mit einem Querschnitt von 0,5 mm^2 verwendet. Die LED-Matten sind über WAGO-Klemmen mit dem Netzteil verbunden. Die Verbindung von Netzteil zu den WAGO-Klemmen wurde mit 2,5 mm^2 Kabeln hergestellt.

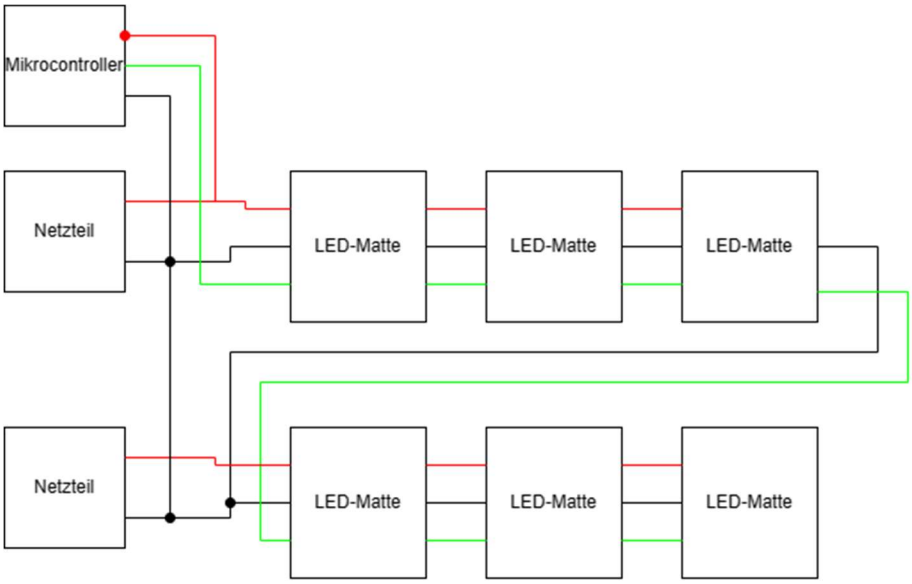

Abbildung 1: Verkabelung der einzelnen LED-Matten (rot = +5 V, schwarz = GND, grün = Data)

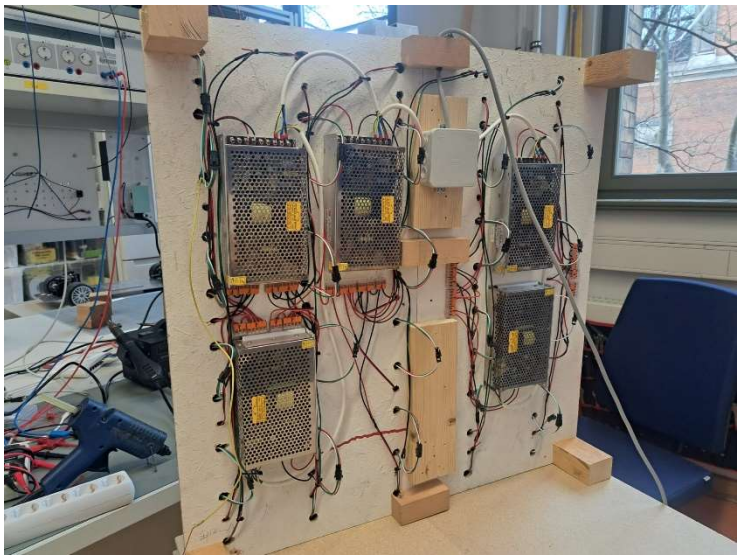

Abbildung 2: Die Verkabelung der LED-Matte mit den verwendeten Netzteilen

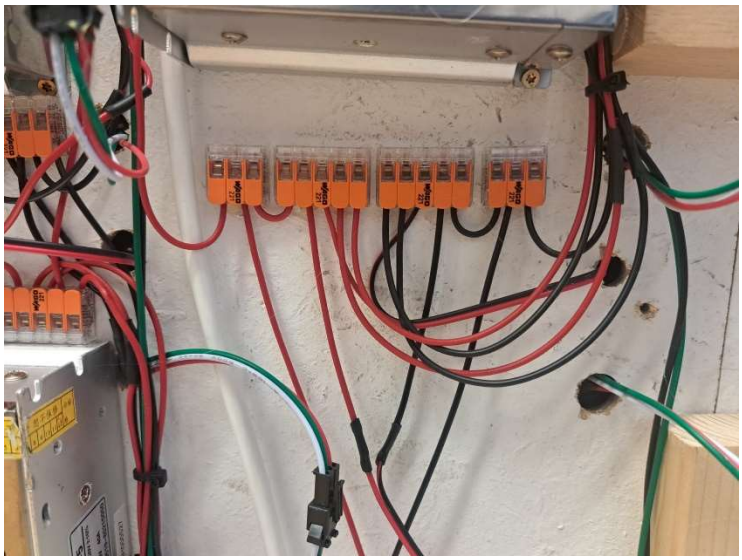

Abbildung 3: Verbindung von 5 Matten mit einem Netzteil

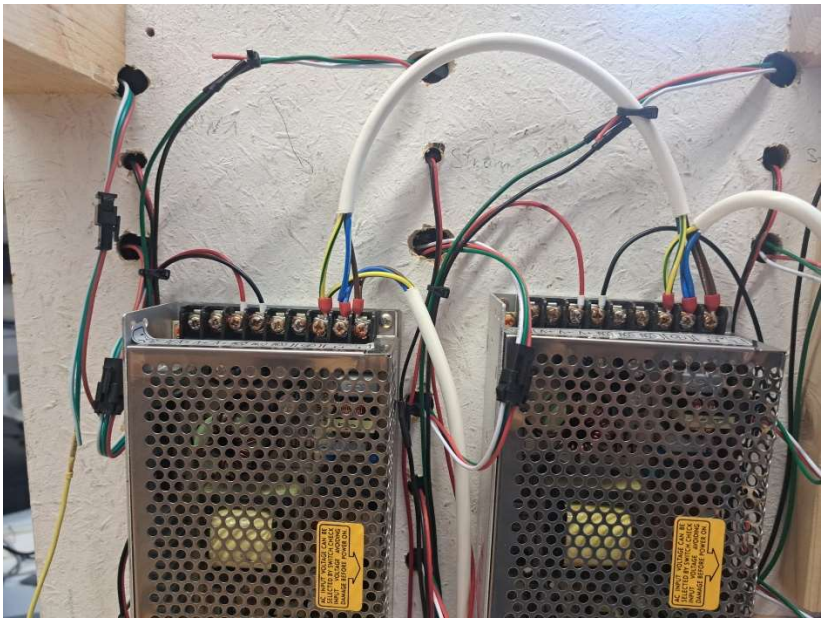

Abbildung 4: Verdrahtung der einzelnen Netzteile. Die Netzteile wurden alle per Daisy-Chain verbunden

3 Balanceboards

Jedes Balanceboard ist mit einem Gyroskop-Sensor (BNO085), einem ESP8266-D1-Mini, einem Akku und zwei LEDs ausgestattet. Der Gyroskopsensor ist ein 9-Achsen-Sensor, der über den I2C-Bus mit dem D1 Mini verbunden ist. Der D1 Mini wurde aufgrund seiner WLAN-Fähigkeit und seiner geringen Energieanforderungen ausgewählt. Die Stromversorgung für den Mikrocontroller und den Gyrosensor wird durch einen 18650-Lithium-Ionen-Akku bereitgestellt.

Die beiden verbauten LEDs dienen als Statusanzeigen. Sie signalisieren, ob das Balanceboard mit der LED-Matte über WLAN verbunden ist und ob der Akkustand des Akkus eine bestimmte Spannung unterschreitet.

Die Software wurde in der Arduino-Sprache entwickelt. Um Daten an das jeweilige Spiel zu senden, muss sich der Mikrocontroller mit dem bereitgestellten Access Point des Spiels verbinden. Hierfür werden die Bibliotheken "ESP8266WiFi.h" und "WiFiUdp.h" benötigt, sowie der Name und das Passwort des Access Points. Das UDP-Protokoll nutzt den Port 8888, da für das Spiel Pong zwei Balanceboards erforderlich sind. Jedes Board hat eine eindeutige ID, die im Programm festgelegt ist.

Quellcode für die WLAN-Einstellungen am Balanceboard:

```
// Wifi
#define WIFI_SSID "Access-Point LED_Spielmatte"
#define WIFI_PASS "LED_Spielmatte"
unsigned int udpPort = 8888;
IPAddress remoteUDPIP(255, 255, 255, 255);
```

Sobald der D1 Mini erfolgreich mit dem WLAN verbunden ist, liest er die Daten des Gyroskopsensors über den I2C-Bus aus. Nur die Daten der x- und y-Achsen werden benötigt. Diese Daten werden im Bereich zwischen -1 und 1 mit einer Schrittweite von 0,01 ausgegeben, wobei eine 1 einer Auslenkung von 90° und eine -1 einer Auslenkung von -90° entspricht.

Die erfassten Sensorwerte werden ausgewertet und an die Spielengine übermittelt. Eine Grenze von ±0,05 wird festgelegt, um geringfügige Auslenkungen vom Nullpunkt zu erkennen. Überschreitet ein Wert pro Achse 0,05, wird eine 1 an das Spiel übermittelt; unterschreitet er -0,05, wird eine -1 übergeben. Liegen die Werte zwischen den Grenzen, wird eine 0 übermittelt. Diese Daten ermöglichen es dem Spiel, die Schlange oder den Schläger bei Pong zu steuern. Die Grenzwerte können angepasst werden, um ein optimales Spielerlebnis zu gewährleisten.

Quellcode zum Auslesen der Werte und Check, ob Grenzwert überschritten bzw. Unterschritten ist:

```
  //Read datas from Gyroskop
  x_coordination = sensorValue.un.gameRotationVector.i;
  y_coordination = sensorValue.un.gameRotationVector.j;

  if (x_coordination > 0.05){
    x_coordination = 1;
  }else{
    if (x_coordination < -0.05)
    {
      x_coordination = -1;
    }else{
      x_coordination = 0;
    }
  }

  if (y_coordination > 0.05){
    y_coordination = 1;
  }else{
    if (y_coordination < -0.05)
    {
      y_coordination = -1;
    }else{
      y_coordination = 0;
    }
  }
```

Ein Beispiel für die übermittelten Daten an das Spiel lautet wie folgt: "g1,1,-1;". Dies bedeutet, dass das Board mit der ID 1 eine positive Neigung in x-Richtung und eine negative Neigung in y-Richtung aufweist. Nach dem Senden der Daten wird eine Pause von 10 Millisekunden eingelegt, um dem Spiel ausreichend Zeit zum Verarbeiten der Daten zu geben.

Quellcode zum Senden der Nachricht an die Spielengine:

```
sendUDPMessage("g"+String(id)+","+String(x_coordination)+","+String(y_coordination)+";");

void sendUDPMessage(String strMessage)
{
  //Array for messages, only in chars sending
  char message[strMessage.length()+1];

  Serial.println("sending length: " + String(strMessage.length()+1));
  strMessage.toCharArray(message, strMessage.length()+1);

  //Send UDP Message
  UDP.beginPacket(remoteUDPIP, udpPort);
  UDP.write(message);
  UDP.endPacket();
  delay(200);
}
```

4 Spiele

Die Spiele wurden jeweils auf einem ESP32 mit der Arduino eigenen C/C++-Sprache programmiert. Der ESP32 wird verwendet, da er eine WLAN-Schnittstelle besitzt. Ebenfalls ist ein ESP32 notwendig, da er genügend Rechenleistung und Speicher aufbringt, um alle 6.400 LEDs über eine Datenleitung anzusteuern. Ein Arduino Uno ist in der Lage mit einem RAM-Speicher von 2 Kilobyte nur 600 solcher LEDs anzusteuern.

Jede „Spielengine“ (Mikrokontroller mit dem jeweiligen Spiel) dient als WLAN Access Point für die Balanceboards. Die Daten von den Balanceboards werden über das UDP-Protokoll empfangen und ausgewertet. Für die Kommunikation via UDP sind die „WiFi.h“ und „AsyncUDP.h“ notwendig. Die Nachrichten werden dabei nur gelesen, wenn eine Nachricht am Port anliegt.

Quellcode zum Empfangen der Nachrichten über das UDP-Protokoll:

```
if (udp.listen(udpPort))
  {
    // Wenn Nachricht an UDP dann speichern
    Serial.print("UDP Listening on IP: ");
    Serial.println(WiFi.localIP());
    udp.onPacket([](AsyncUDPPacket packet)
                 { udp_message = (const char *)packet.data(); });
  }
```

Die empfangenen Daten werden danach ausgewertet und für den entsprechenden Spielern zugewiesen. Des Weiteren werden die Daten des Gyrosensors analysiert und entsprechend aufbereitet in welche Richtung sich der Schläger oder die Schlange bewegen soll.

Quellcode zur Aufbereitung der empfangenen Daten bei dem Spiel Snake:

```
void breakString(String output)
{
  // lokale Variablen
  String id_gyro;

  // String von Gyrosensor zerteilen
  id_gyro = output.substring(0, output.indexOf(','));
  output = output.substring(output.indexOf(',') + 1, output.length() + 1);
  gyro.x = output.substring(0, output.indexOf(',')).toFloat();
  output = output.substring(output.indexOf(',') + 1, output.length() + 1);
  gyro.y = output.substring(0, output.indexOf(';')).toFloat();
  output = output.substring(output.indexOf(';') + 1, output.length());

  Serial.println("ID:" + id_gyro + ", X: " + String(gyro.x) + ", Y: " + String(gyro.y));

  // Richtung von Schlange bestimmen
  if (gyro.x == 1)
  {
    direction = RIGHT;
  }
  if (gyro.x == -1)
  {
    direction = LEFT;
  }
  if (gyro.y == -1)
  {
    direction = UP;
  }
  if (gyro.y == 1)
  {
    direction = DOWN;
```

```
  }
  if (gyro.x == 0 && gyro.y == 0)
  {
    direction = NONE;
  }
}
```

Die Kommunikation zwischen LED-Matte und Spiel funktioniert bei allen Spielen gleich. Es muss beachtet werden, dass obwohl die Matte aus 80 x 80 LEDs (oder auch Pixel bezeichnet) besteht, sind die 25 einzelnen LED-Matten über die Datenleitung in Reihe geschalten. Somit entsteht keine 80 x 80 Pixel große LED-Matte, sondern eine 16 x 400 Pixel Matte. Zur Ansteuerung der einzelnen LEDs wird die Bibliothek „FastLED.h“ verwendet, da diese im Gegensatz zu „Neopixel.h“ mehr Vorteile bietet. Ein Beispiel hierfür ist die zeilenweise Ansteuerung der einzelnen LEDs. Zu dem bietet diese Bibliothek ein vorgefertigtes Beispiel zur Ansteuerung einer 16 x 16 LED-Matte. Dieses Beispiel wurde für die einzelnen Spiele verwendet und den gegebenen Anforderungen angepasst.

Durch eine Umrechnung der Koordinaten im Quellcode kann die Größe der Matte beliebig eingestellt werden und jede LED mit den richtigen Koordinaten angesprochen werden. Somit ist die Größe von 80 x 80 wieder hergestellt.

Quellcode zur Umrechnung der Größe der LED-Matte:

```
// Umrechnung der Koordinaten in Koordinaten für Matrix
int x = x_coord_led % heigth_one_led_mat;
int y = Y_MAX * (x_coord_led / heigth_one_led_mat) + y_coord_led;
```

Hierbei wird der Funktion der Wert für x_coord_led und y_coord_led übergeben, welche die Koordinaten für auf der LED-Matte entsprechen. Height_one_led_mat ist die Höhe einer einzelnen LED-Matte. In diesem Fall ist der Wert 16. Y_MAX ist hier die Gesamthöhe der LED-Matte in Pixel bzw. LEDs.

Durch die Umrechnung im Quellcode können auch kleinere LED-Matten mit den Spielen betrieben werden. Dazu aber mehr im Abschnitt Nachbau.

Wenn das jeweilige Spiel gestartet oder durch ein Game Over beendet wird, so startet das Spiel automatisch neu. Falls ein Game Over erfolgen sollte, so wird auf der Matte das Wort „Game Over“ angezeigt.

Bei einem Neustart wird am Anfang des Spiels die Spielfläche initialisiert. Dabei werden die Spielfeldumrandung sowie die Startpositionen der Schlange bzw. der Schläger gezeichnet. Hierfür wurden im Quellcode die zwei Funktionen initalizeGame() und drawingField() angelegt.

Um die einzelnen LEDs im Quellcode anzusprechen, muss an die Funktion sendData() die Koordinaten der jeweiligen Änderung sowie die dazugehörige Farbe übermittelt werden. Wenn die LEDs ausgeschaltete werden sollen, so muss die Farbe Schwarz übergeben werden. Ebenfalls in dieser Funktion wird die Umrechnung der Koordinaten, welche oben beschrieben wurde, durchgeführt.

Aufruf der Funktion sendData():

```
sendData(food.x, food.y, food.status);
```

4.1.1 Pong

Pong ist ein 2 Personen Spiel, bei dem die Spieler jeweils einen Schläger steuern und versuchen müssen den Ball nicht hinter den eigenen Schläger zu bekommen. Sollte ein Ball hinter einen Schläger des Gegners gelangen, erhält der Spieler einen Punkt.

Für Pong wurde eine Spielfeldbegrenzung eingerichtet, welche jeweils auf der linken und rechten Seite der Matte eine Spalte beträgt. Die Farbe des Spielfeldrandes ist weiß.

Die Schläger sind jeweils 2 Pixel hoch und 16 Pixel breit. Per Balanceboard werden die Schläger jeweils nach links oder rechts gesteuert. Die Farbe der Schläger ist auf Rot eingestellt.

Quellcode zur Bewegung der Schläger:

```
void movePaddle()
{
  // lokale Variablen
  int step = 2;

  // Paddle1 nach rechts bewegen
  if (directionPaddle1 == RIGHT && paddle1[paddleLength - 1].x != X_MAX - 1 - step)
  {
    // letzten LEDs auschalten
    for (int i = 0; i < step; i++)
    {
      sendData(paddle1[i].x, paddle1[0].y, BLACK);
      sendData(paddle1[i].x, paddle1[1].y, BLACK);
    }

    // Paddle1 bewegen
    for (int i = 0; i < paddleLength; i++)
    {
      paddle1[i].x = paddle1[i].x + step;
      sendData(paddle1[i].x, paddle1[0].y, RED);
      sendData(paddle1[i].x, paddle1[1].y, RED);
    }
  }
  // Paddle1 nach links bewegen
  if (directionPaddle1 == LEFT && paddle1[0].x != step)
  {
    // letzten LEDs auschalten
    for (int i = 1; i < step + 1; i++)
    {
      sendData(paddle1[paddleLength - i].x, paddle1[0].y, BLACK);
      sendData(paddle1[paddleLength - i].x, paddle1[1].y, BLACK);
    }

    // Paddle1 bewegen
    for (int i = 0; i < paddleLength; i++)
    {
      paddle1[i].x = paddle1[i].x - step;
      sendData(paddle1[i].x, paddle1[0].y, RED);
      sendData(paddle1[i].x, paddle1[1].y, RED);
    }
  }

  // Paddle2 nach rechts bewegen
  if (directionPaddle2 == RIGHT && paddle2[paddleLength - 1].x != X_MAX - 1 - step)
  {
    // letzten LEDs auschalten
    for (int i = 0; i < step; i++)
    {
      sendData(paddle2[i].x, paddle2[0].y, BLACK);
      sendData(paddle2[i].x, paddle2[1].y, BLACK);
    }
```

```
    // Paddle2 bewegen
    for (int i = 0; i < paddleLength; i++)
    {
      paddle2[i].x = paddle2[i].x + step;
      sendData(paddle2[i].x, paddle2[0].y, RED);
      sendData(paddle2[i].x, paddle2[1].y, RED);
    }
  }

  // Paddle2 nach links bewegen
  if (directionPaddle2 == LEFT && paddle2[0].x != step)
  {
    // letzten LEDs auschalten
    for (int i = 1; i < step + 1; i++)
    {
      sendData(paddle2[paddleLength - i].x, paddle2[0].y, BLACK);
      sendData(paddle2[paddleLength - i].x, paddle2[1].y, BLACK);
    }

    // Paddle2 bewegen
    for (int i = 0; i < paddleLength; i++)
    {
      paddle2[i].x = paddle2[i].x - step;
      sendData(paddle2[i].x, paddle2[0].y, RED);
      sendData(paddle2[i].x, paddle2[1].y, RED);
    }
  }
}
```

Der Ball ist quadratisch und besitzt somit in der Höhe und der Breite 2 Pixel. Die Farbe wurde auf Grün eingestellt. Die Bewegung des Balls erfolgt immer in eine X und in eine Y-Richtung. Prallt der Ball gegen einen Schläger oder gegen die Spielfeldbegrenzung, so ändert sich die Richtung des Balls nur auf einer Achse. Ein Beispiel: Der Ball bewegt sich nach rechts oben. Wenn der Ball gegen eine Seitenwand prallt, ändert sich die Bewegung zu links oben.

Quellcode zur Bewegung des Balls:

```
void moveBall()
{
  // lokale Variablen
  int directionX;
  int directionY;
  int step = 1;

  // alte Ballposition ausschalten (LEDs)
  for (int i = 0; i < ballLength; i++)
  {
    for (int j = 0; j < ballLength; j++)
    {
      sendData(ball[i].x, ball[j].y, BLACK);
    }
  }

  // Richtung des Balls definieren
  switch (directionBall)
  {
  case RIGHT_UP:
    directionX = step;
    directionY = step;
    break;

  case RIGHT_DOWN:
    directionX = step;
```

```
    directionY = -step;
    break;

  case LEFT_UP:
    directionX = -step;
    directionY = step;
    break;

  case LEFT_DOWN:
    directionX = -step;
    directionY = -step;
    break;
  }

  // neue Ball Position berechnen
  for (int i = 0; i < ballLength; i++)
  {
    ball[i].x = ball[i].x + directionX;
    ball[i].y = ball[i].y + directionY;
  }

  // neue Ball Position einschalten (LEDs)
  for (int i = 0; i < ballLength; i++)
  {
    for (int j = 0; j < ballLength; j++)
    {
      sendData(ball[i].x, ball[j].y, GREEN);
    }
  }
}
```

Quellcode zum Prüfen, ob der Ball gegen die Wand oder einen Schläger geprallt ist:

```
void checkCollision()
{
  // Check ob Ball gegen Wand geprallt ist
  if ((ball[0].x < 2) || (ball[1].x > X_MAX-3))
  {
    switch (directionBall)
    {
    case RIGHT_UP:
      directionBall = LEFT_UP;
      break;
    case RIGHT_DOWN:
      directionBall = LEFT_DOWN;
      break;
    case LEFT_UP:
      directionBall = RIGHT_UP;
      break;
    case LEFT_DOWN:
      directionBall = RIGHT_DOWN;
      break;
    }
  }

  // Check ob Ball gegen Paddle1 geprallt ist
  for (int i = 0; i < paddleLength - 1; i++)
  {
    if (ball[0].y <= paddle1[1].y + 1 && ball[0].x == paddle1[i].x)
    {
      switch (directionBall)
      {
      case LEFT_DOWN:
        directionBall = LEFT_UP;
        break;

      case RIGHT_DOWN:
```

```
        directionBall = RIGHT_UP;
        break;
      }
    }
  }

  // Check ob Ball gegen Paddle2 geprallt ist
  for (int i = 0; i < paddleLength; i++)
  {
    if (ball[1].y >= paddle2[0].y - 1 && ball[0].x == paddle2[i].x)
    {
      switch (directionBall)
      {
      case LEFT_UP:
        directionBall = LEFT_DOWN;
        break;

      case RIGHT_UP:
        directionBall = RIGHT_DOWN;
        break;
      }
    }
  }

  // Score erhöhen nache Punkteerfolg
  if (ball[0].y < paddle1[0].y)
  {
    score2++;
    stateGame = SCORE;
  }

  if (ball[1].y > paddle2[1].y)
  {
    score1++;
    stateGame = SCORE;
  }
}
```

Um ein faires Spiel zu realisieren, müssen beide Spieler nach einem Punktgewinn oder einem Neustart des Spiels zuerst mit ihrem Balanceboard nach hinten kippen. Erst dann wird der Ball durch das Spiel freigegeben. Die Ballbewegung und die Position des Balls werden am Anfang des Spiels per Zufall entschieden, so dass keine Vorteile durch einen Spieler entstehen können.

4.1.2 Snake

In dem Spiel Snake muss eine Schlange gesteuert werden. Am Anfang ist die Schalenge sehr klein. Sobald ein Apfel gefressen wird, wächst die Schlange um eine vorgegebene Länge. Das Spiel endet sofort, wenn sich die Schlange selbst frisst oder der Spielfeldrand berührt wird.

In diesem Spiel ist die Schlange zu Beginn des Spiels eine LED groß. Wenn ein Apfel, welcher durch eine grüne LED symbolisiert wird, gefressen wird so wächst die Schlage um eine LED. Der Kopf der Schlange wird bei diesem Spiel in Rot dargestellt, um einen eindeutigen Beginn der Schlange festzulegen. Der restliche Körper der Schlange wird im späteren Verlauf in Orange dargestellt.

Falls der Apfel erfolgreich gefressen wurde, so wird ein neuer Apfel auf der Matte per Zufall generiert. Der Apfel kann dabei aber nicht auf der Schlange erscheinen.

Quellcode zur Erstellung des Apfels in der Funktion:

```
void spawnFood()
{
  // Position von Essen erzeugen
  food.x = random(2, X_MAX - 2);
  food.y = random(2, Y_MAX - 2);
  food.status = 2;

  // Sicherstellen das Essen nicht auf Schlange spawnt
  for (int i = 0; i < snakeLength; i++)
  {
    if (food.x == snake[i].x && food.y == snake[i].y)
    {
      spawnFood();
      return;
    }
  }

  // Essen auf Matrix dartstellen
  sendData(food.x, food.y, food.status);
}
```

Durch das Verbinden des Spiels mit einem Balanceboard, kann die Schlange in vier Richtungen gesteuert werden, um zu dem vorgegebenen Ziel zu gelangen. Die maximale Länge der Schlange kann im Quellcode eingestellt werden. In diesem Projekt ist eine Länge von maximal 100 LEDs eingestellt.

Wenn die Schlange sich bewegt, so muss die erste LED in Bewegungsrichtung eingeschalten werden und die letzte LED der Schlange ausgeschalten werden.

Quellcode zur Bewegung der Schlange und Definition des Kopfes:

```
void moveSnake()
{
  Coordinations head;

    led.x = snake[snakeLength - 1].x;
    led.y = snake[snakeLength - 1].y;

  head = snake[0];
  switch (direction)
      {
      case UP:
        head.y--;
        break;
      case DOWN:
```

```
        head.y++;
        break;
      case LEFT:
        head.x--;
        break;
      case RIGHT:
        head.x++;
        break;
      }

  if (checkSnakeCollision(head)){
    return;
  }

  for (int i = snakeLength - 1; i > 0; i--)
  {
    snake[i].x = snake[i - 1].x;
    snake[i].y = snake[i - 1].y;
    snake[i].status = snake[i - 1].status;
    sendData(snake[i].x, snake[i].y, snake[i].status);
  }

  snake[0] = head;

  sendData(snake[0].x, snake[0].y, RED);
  sendData(led.x, led.y, BLACK);
}
```

Eine Besonderheit bei Snake ist, wenn die Balance auf dem Balanceboard gehalten wird, bewegt sich die Schlange nicht weiter.

Quellcode zur Bestimmung der Richtung der Schlange:

```
  // Richtung von Schlange bestimmen
  if (gyro.x == 1)
  {
    direction = RIGHT;
  }
  if (gyro.x == -1)
  {
    direction = LEFT;
  }
  if (gyro.y == -1)
  {
    direction = UP;
  }
  if (gyro.y == 1)
  {
    direction = DOWN;
  }
  if (gyro.x == 0 && gyro.y == 0)
  {
    direction = NONE;
  }
```

5 Probleme bei der Umsetzung und Verbesserungen

Ein Problem bei der Umsetzung, ist die Anzahl der benötigten Netzteile. Durch ein geschriebenes Testprogramm und einem Leistungsmesser an der Matte konnte die Leistung in Abhängigkeit der Helligkeit der Matte festgestellt werden. Die Werte sind in nachfolgender Tabelle aufgeführt.

Digitalwert der Helligkeit (0 – 255)	Benötigte Leistung (Watt)
10	46,8
20	94
30	143
40	193
50	240
60	288
70	334
80	380
90	426
100	468
110	511
120	550
130	584
141	616
150	643
160	670
170	690
180	730
190	750
200	763
210	775
220	790
230	800
240	806
251	813

Man sieht das die fünf Netzteile die volle Helligkeit der LEDs abfangen können. Da bei einer solchen enormen Helligkeit keine Spiele auf dieser Matte spielbar sind, würde es sich anbieten die Helligkeit bei den Spielen auf einen Digitalwert von 25 oder kleiner einzustellen. Bei dieser Helligkeit ist in Verbindung mit einer Milchglasplexiglasplatte, welche auf den LEDs aufliegen sollte, ist ein direktes spielen möglich, da das Licht in den Augen nicht blendet und sehr angenehm ist. Bei dieser Helligkeit können somit auch drei Netzteile eingespart werden. Des Weiteren werden bei den Spielen nie alle LEDs eingeschalten. So liegt der Maximalverbrauch der Matte bei dem Spiel Pong bei maximal 60 W. In diesem Fall wäre nur noch ein Netzteil nötig und könnte ebenfalls kleiner dimensioniert werden. Beispielsweise auf 100 W.

Ein weiteres Problem bei der Umsetzung ist eine große Zeitverzögerung zwischen der Bewegung auf dem Balanceboard und der Anzeige der Bewegung auf der LED-Matte. Die Verzögerung beträgt circa 2 Sekunden, welche durch die Nutzung nur einer Datenleitung entsteht. Diese Datenleitung muss immer 6.400 LEDs ansprechen. Durch eine Parallelisierung der Datenleitung auf beispielsweise 5 Leitungen könnte die Verzögerung deutlich minimiert werden.

Für einen solchen Fall müsse der Quellcode dementsprechend angepasst werden. Durch mathematische Berechnungen muss bestimmt werden, wann ein Übergang zwischen den LED-

Matten stattfindet, und eine Umrechnung der Koordinaten müsste bei einem solchen Übergang erfolgen.

6 Nachbau

Diese LED-Matte kann auch nachgebaut werden. Hierbei kann die Größe der Matte nach belieben festgelegt werden. Es muss allerdings beachtet werden, dass bei einer größeren oder kleineren Matte der Quellcode der Spiele demzufolge angeglichen werden muss. Ebenfalls müssen die Netzteile der Größe der Matte und der benötigten Leistung angepasst werden.

In diesem vorgestellten Falle werden benötigt:

Anzahl	Bauteile
25	LED-Matten mit 16 x 16 NEO-Pixel RGB LEDS mit dem Typ WS2812B
2 bis 5	Netzteile, da die Helligkeit nach Belieben eingestellt werden kann.
2	ESP32 DEV-Module für die jeweiligen Spiele
2	ESP8266 D1 Mini für die Balanceboards
2	BNO085 Gyroskopsensoren
1	Kartenhalter für die jeweiligen Spiele

Des Weiteren werden diverse Widerstände, LEDs und Schalter für die Balanceboards benötigt. Die genaue Anzahl und Bezeichnung können aus den Schaltplänen entnommen werden. Ebenso werden genügend Kabel, welche ausreichend dimensioniert werden sollten für die Verkabelung der LED-Matte benötigt, da die angelöteten Kabel eventuell verlängert werden müssen. Zusätzlich ist ein Brett nötig auf dem die Matten aufgebracht werden können. Bei der Verkabelung der Matten ist zu beachten, dass das Datenkabel und das Massekabel der alle LED-Matten verbinden muss. Wenn eine Reihe fertig verdrahtet, ist so muss die Daten- und Masseleitung an den Anfang der nächsten Reihe angeklemmt werden. Siehe dazu Abbildung 1.

Im Quellcode der Balancebords muss für einen Nachbau nichts angepasst werden. Im Quellcode der Spiele muss die Größe der Matte eingestellt werden. Hierbei gibt es jeweils für die X und für die Y-Koordinaten eine Größe. Des Weiteren können die Größe der Schläger bei Pong sowie des Balls angepasst werden und die maximale Länge der Schlange bei Snake. Ebenfalls können die Farben für die Objekte im Spiel umgestellt werden. Hierzu empfiehlt es sich die Dokumentation der Bibliothek „FastLED.h“ genauer zu lesen.

Die Änderungen des Spiels können in folgenden Zeilen des Quellcodes eingestellt werden. Diese befinden sich am Anfang des Quellcodedokuments.

Für Pong:

```
//--------------------------------------------------------------------------------
// Einstellungen, welche vorgenommen werden können
#define heigth_one_led_mat 16     // Höhe einer einzelnen verbauten LED-Matte
#define count_led_mat 25          // Anzahl der verbauten LED-Matten
#define x_coordinations_length 80 // Anzahl der LEDs auf der x-Achse
#define y_coordinations_length 80 // Anzahl der LEDs auf der y-Achse
#define brightness_led 25         // Helligkeit der Matte einstellen
#define paddle_length 16          // Länge der Schläger
#define paddle_heigth 2           // Höhe der Schläger
#define ball_length 16            // Größe des Balls
#define max_score 5               // Maximal erreichbarer Score
//--------------------------------------------------------------------------------
```

Für Snake:

```
//--------------------------------------------------------------------------------
// Einstellungen, welche vorgenommen werden können
#define heigth_one_led_mat      16  // Höhe einer einzelnen verbauten LED-Matte
#define count_led_mat           25  // Anzahl der verbauten LED-Matten
#define x_coordinations_length  80  // Anzahl der LEDs auf der x-Achse
#define y_coordinations_length  80  // Anzahl der LEDs auf der y-Achse
#define brightness_led          25  // Helligkeit der Matte einstellen
#define snake_length            100 // Länge der Schlange einstellen
//--------------------------------------------------------------------------------
```