Snake

anhand eines JOY-iT Mega und einer LED-Matrix

Maik Karimi, Niclas Kober TES21

Motivation

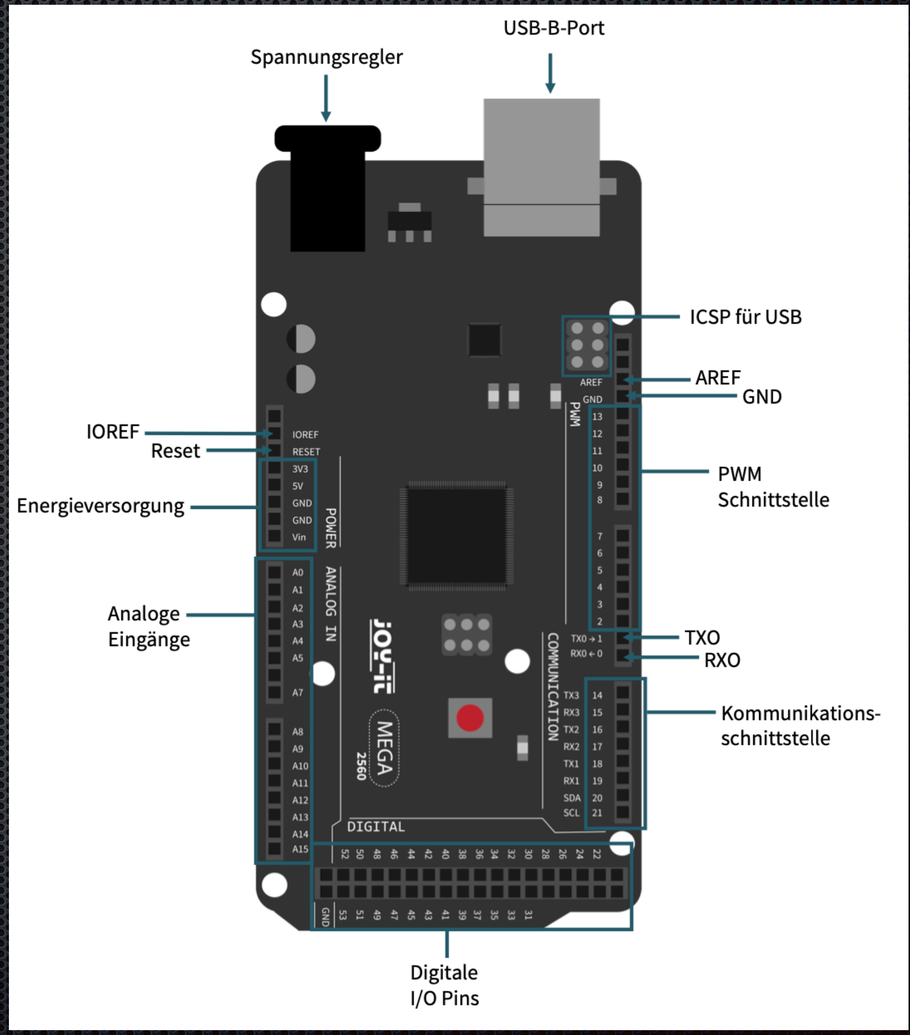
- Klassisches Snake mit geeigneter Hard- und Software umsetzen
- 2 LED Matrizen für größere Spielfläche
- Steuerung mithilfe zweier Buttons

Gliederung

- Verwendete Hardware
- Verkabelung / Verschaltung
- Code

Hardware

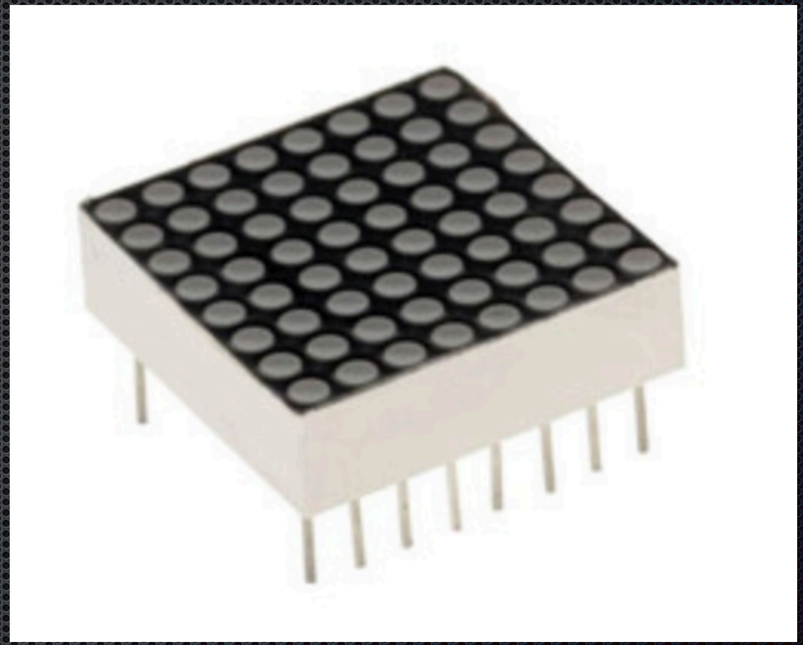
- JOY-iT MEGA 2560



Quelle: Link

Hardware

- 8 X 8 LED-Matrix



Quelle: Link

Hardware

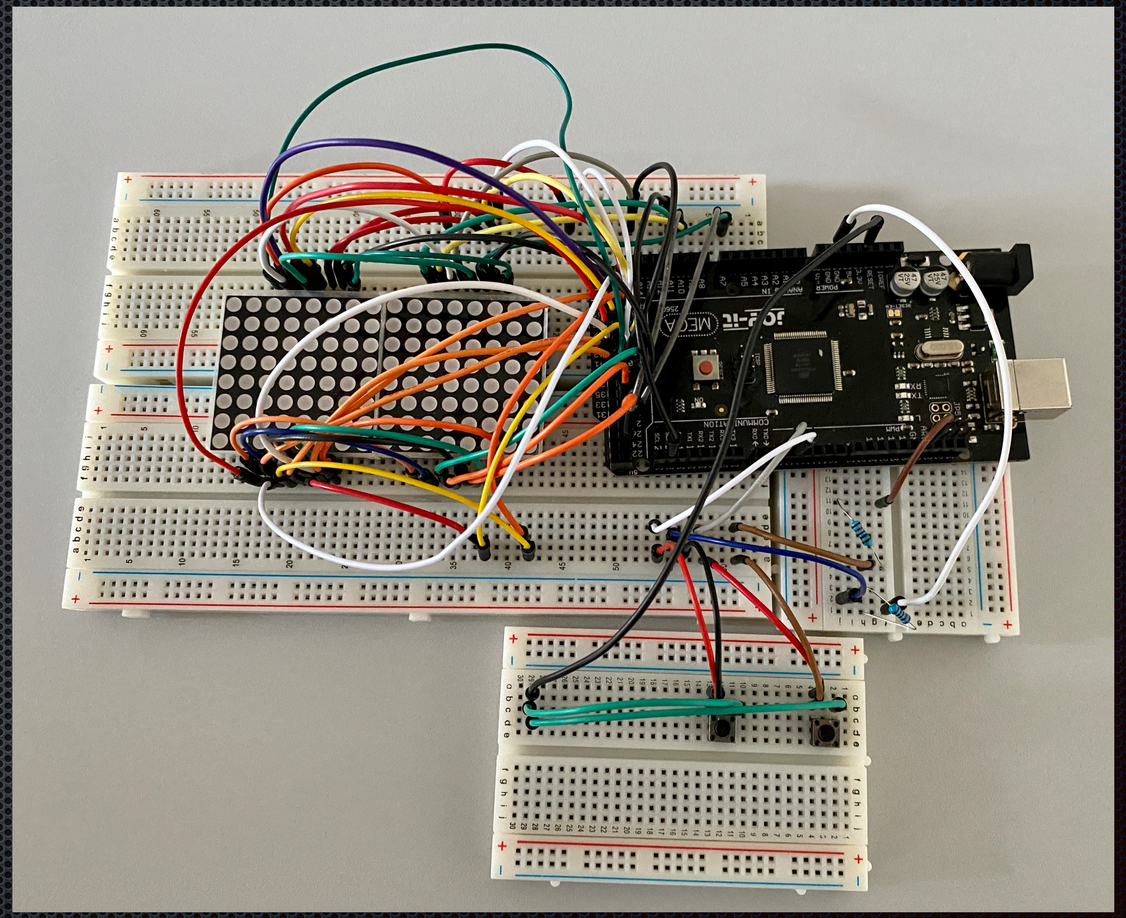
- Taster und Widerstände

Quelle: Link

Quelle: Link

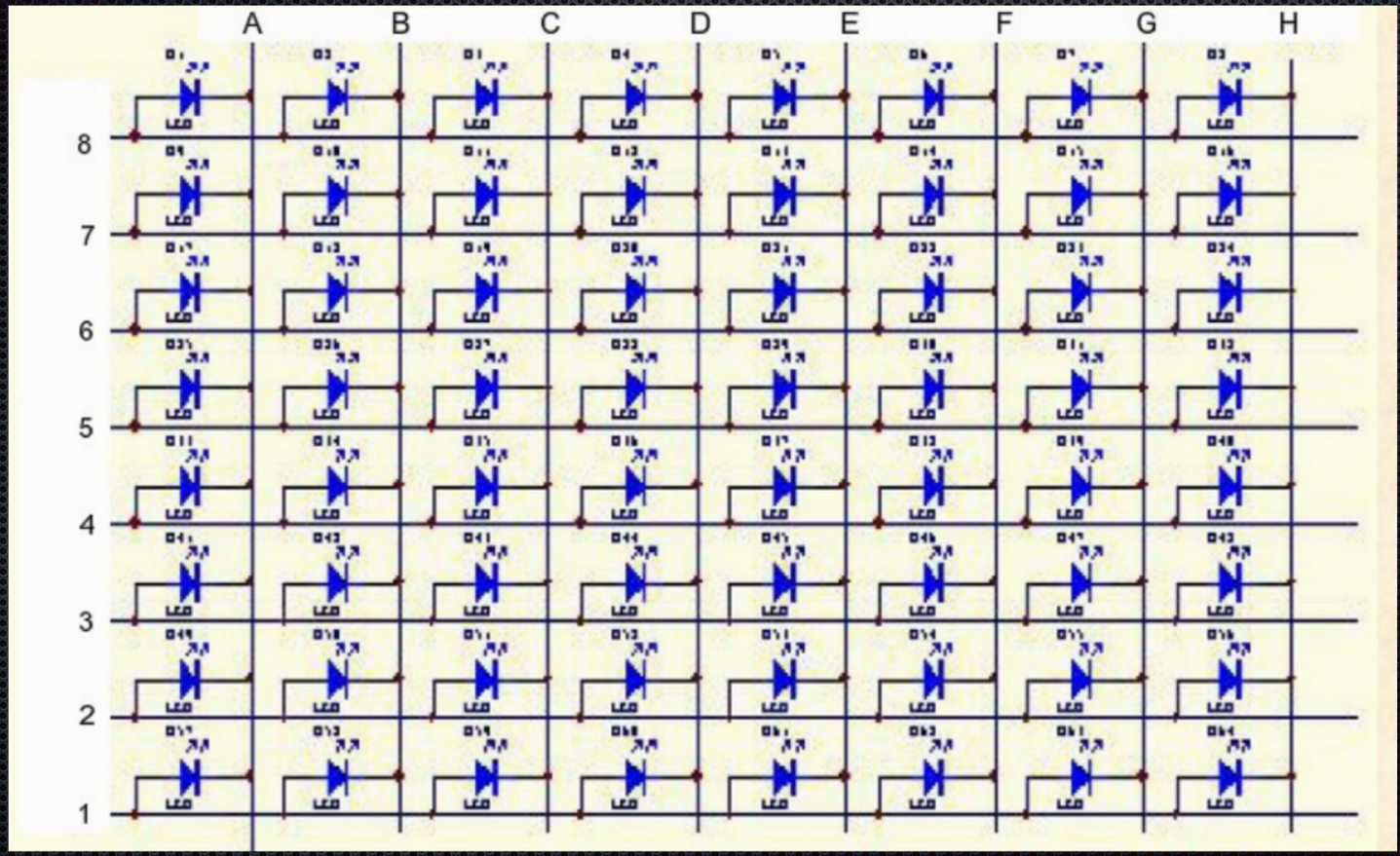
Hardware

- Finales Projekt



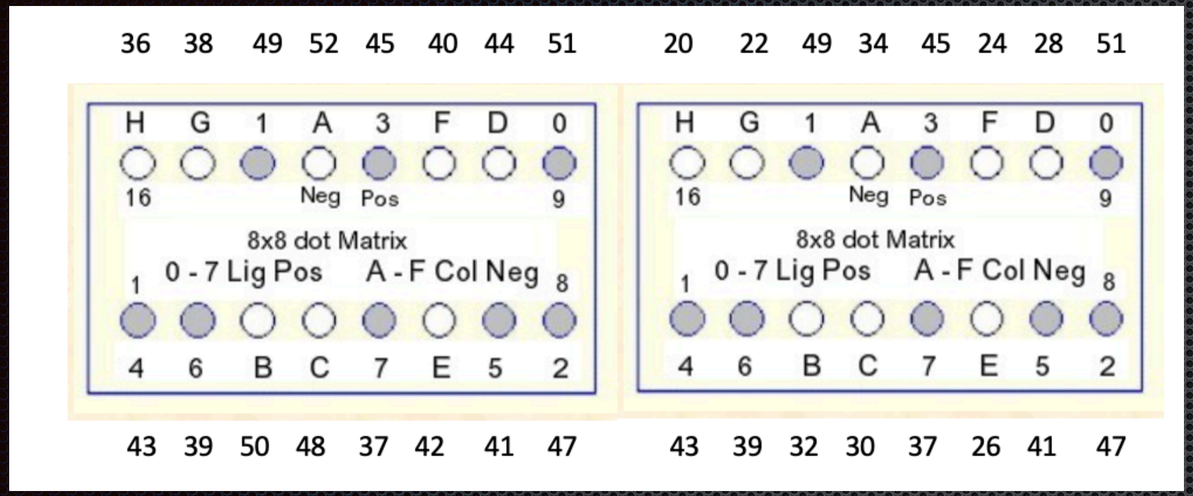
Quelle: Eigene Aufnahme

Schaltplan der Matrix



Quelle: Link

Pin - Verdrahtung mit dem Board



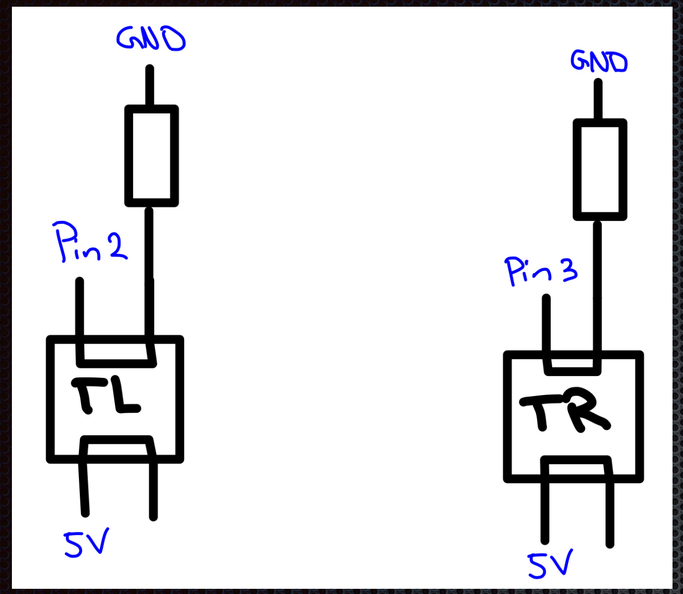
Quelle: Link

Quelle: Link

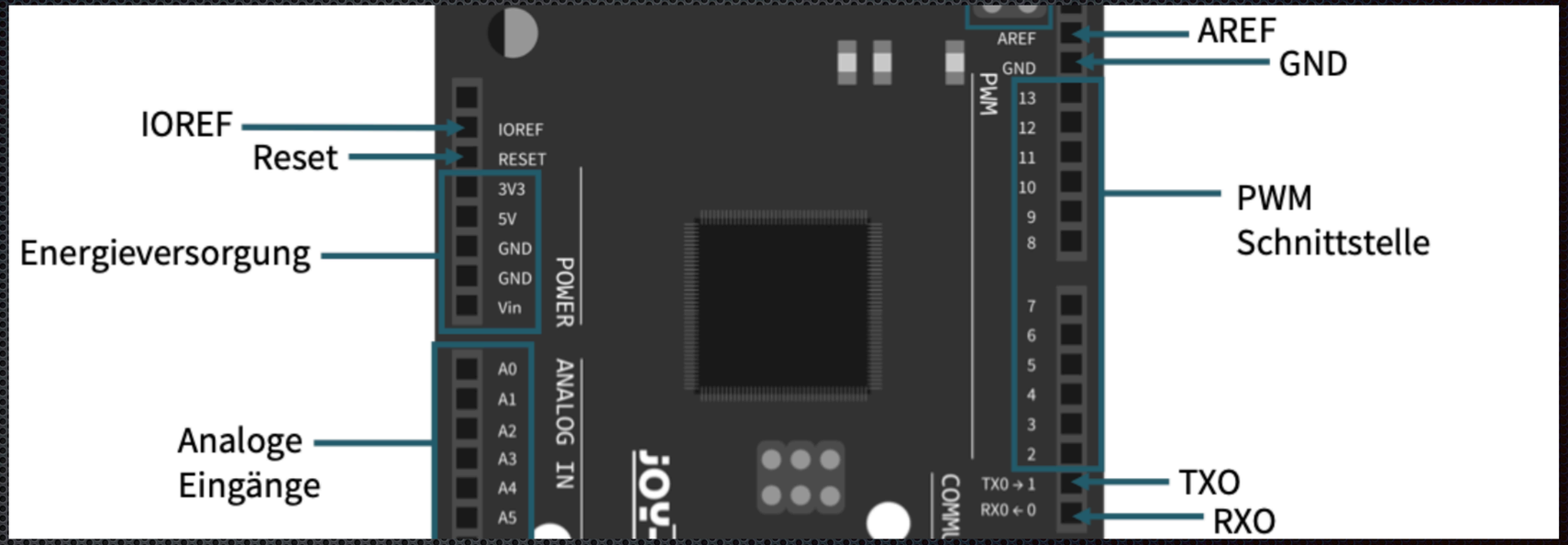
—> Bei Buchstaben: 0 (neg) = an

—> Bei Zahlen : 1 (pos) = an

Verschaltung der Taster



Quelle: Eigene Aufnahme



Quelle: Link

Definitionen & Variablen

```
#define RECHTS 0
#define UNTEN  1
#define LINKS  2
#define OBEN   3

typedef struct {      // struct with x and y coordinate for one element of the LED matrix
  int x;
  int y;
} point;

point points[128];  // the Snake is an array of LED points
point fruit;
int direction;
int size;
int movecount;
int hitMutex;
int turnMutex;
int Speed = 450;
void Init();
void Buttons();
void ButtonR();
```

setup() - Funktion

```
void setup() {
  // put your setup code here, to run once:
  pinMode(2, INPUT);                    // Button pin Left
  pinMode(3, INPUT);                    // Button pin Right
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  // connects the interrupt pins with the interrupt handler
  attachInterrupt(digitalPinToInterrupt(3), my_interrupt_handlerR, RISING);
  attachInterrupt(digitalPinToInterrupt(2), my_interrupt_handlerL, RISING);
```

setup() - Funktion

```
  for(int i = 0; i<17;i++){      // Pin Setup as output for all pins
    pinMode(52-(2*i), OUTPUT);
    pinMode(51-(2*i), OUTPUT);
  }
  for(int i = 0;i<8;i++){
    digitalWrite(51-(2*i), 0);     // All LEDs off
  }
  for(int i = 0;i<16;i++){
    if(i>=3){
      digitalWrite(52-(2*i)-2, 1);  // All LEDs off but they are negated so off means "1",
                                    // because of the pin setup, the pins from 3 up have to be one position further
    } else {
    digitalWrite(52-(2*i), 1);
    }
  }
  randomSeed(analogRead(0));      // Random spot in the equal sequence of random numbers for the fruit
Init();
movecount=millis();
}
```

Init() - Funktion

```
void Init(){                          // restart the game

size=3;                               // limits the array so we only check the elements that the snake actually has
for(int i = 0;i<128;i++){             // snake elements initialized outside of the LED matrix
  points[i].x = 100;
  points[i].y = 100;
  }
  for (int i = 0;i<size;i++){         // start position for the snake in the middle of the field
    points[i].x= 8-i;
    points[i].y= 4;
  }
  Fruit();
  direction=RECHTS;
}
```

Fruit() - Funktion

```
void Fruit(){

fruit.x = random(1,17);                    // Random coordinates for the fruit
fruit.y = random(1,9);

for(int i = 0; i<size;i++){
  if((fruit.x == points[i].x)&&(fruit.y == points[i].y)){     // if fruit spawns on the snake -> try again
    Fruit();
    break;
    }
  }
}
```

loop() - Funktion

```
void loop() {
 LED();
  move();
 if(hitMutex){
  Hit();
 }
 fail();
 speed();
}
```

LED() - Funktion

```
void LED(){

  for(int i = 0;i<size;i++){
    if(points[i].x >=4){



    digitalWrite(54-(points[i].x*2)-2,0);
    digitalWrite(53-points[i].y*2,1);

    digitalWrite(54-(points[i].x*2)-2,1);
    digitalWrite(53-points[i].y*2,0);

}else {

    digitalWrite(54-points[i].x*2,0);
    digitalWrite(53-points[i].y*2,1);

    digitalWrite(54-points[i].x*2,1);
    digitalWrite(53-points[i].y*2,0);
}
```

```
if(fruit.x>=4){
    digitalWrite(54-(fruit.x*2)-2,0);
    digitalWrite(53-fruit.y*2,1);

    digitalWrite(54-(fruit.x*2)-2,1);
    digitalWrite(53-fruit.y*2,0);
} else {

    digitalWrite(54-fruit.x*2,0);
    digitalWrite(53-fruit.y*2,1);

    digitalWrite(54-fruit.x*2,1);
    digitalWrite(53-fruit.y*2,0);


    }
  }
}
```

loop() - Funktion

```
void loop() {
 LED();
  move();
 if(hitMutex){
  Hit();
 }
 fail();
 speed();
}
```

move() - Funktion

```
void move() {
  int time = millis();
if((time-movecount)>=Speed){    // move only triggers every (speed) milliseconds
  movecount=millis();
  hitMutex = 1;                 // hit may activate again
  turnMutex = 1;                // direction may change again
```

move() - Funktion

```
switch(direction){                  // every element of the snake gets the coordinates of the next one
                                    // and the first one changes according to the movement direction
```

```
case OBEN:
  for(int  i = size-1; i>0; i--){
  points[i] = points[i-1];
  }
  points[0].y++;
  Positioning();
  break;
```

```
case LINKS:
  for(int  i = size-1; i>0; i--){
  points[i] = points[i-1];
  }
  points[0].x--;
  Positioning();
  break;
```

```
case RECHTS:
  for(int  i = size-1; i>0; i--){
     points[i] = points[i-1];
    }
  points[0].x=points[0].x+1;
  Positioning();
  break;
```

```
case UNTEN:
  for(int  i = size-1; i>0; i--){
  points[i] = points[i-1];
  }
  points[0].y--;
  Positioning();
  break;
```

Positioning() - Funktion

```
void Positioning(){
  for(int i = 0; i<size;i++){

    if(points[i].x>16){
      points[i].x = points[i].x-16;
    }
    if(points[i].x<1){
      points[i].x = points[i].x+16;
    }
    if(points[i].y>8){
      points[i].y = points[i].y-8;
    }
    if(points[i].y<1){
      points[i].y = points[i].y+8;
    }
  }
}
```

loop() - Funktion

```
void loop() {
 LED();
  move();
 if(hitMutex){
  Hit();
 }
 fail();
 speed();
}
```

Hit() - Funktion

```
void Hit(){
  if((points[0].x == fruit.x)&&(points[0].y == fruit.y)){       // head of the snake hits the fruit
    size++;
    hitMutex=0;                                                 // hit should not be called more than once between moves
```

```
case OBEN:
  points[size-1].x = points[size-2].x;
  points[size-1].y=points[size-2].y-1;
  break;
```

```
case LINKS:
  points[size-1].y = points[size-2].y;
  points[size-1].x=points[size-2].x+1;
  break;
```

```
switch(direction){
  case RECHTS:
    points[size-1].y = points[size-2].y;
    points[size-1].x=points[size-2].x-1;
    break;
```

```
case UNTEN:
  points[size-1].x = points[size-2].x;
  points[size-1].y=points[size-2].y+1;
  break;
```

```
Fruit();        // Fruit is hit, so we need a new one
```

loop() - Funktion

```
void loop() {
 LED();
  move();
 if(hitMutex){
  Hit();
 }
 fail();
 speed();
}
```

fail() - Funktion

```
void fail(){
  for(int i = 1; i<size;i++){
    if((points[0].x==points[i].x)&&(points[0].y == points[i].y)){
      Init();
    }
  }
}
```

loop() - Funktion

```
void loop() {
 LED();
  move();
 if(hitMutex){
  Hit();
 }
 fail();
 speed();
}
```

speed() - Funktion

```
void speed(){
  if(Speed>200){
    Speed = 465 - size*5;
   }
}
```

my_interrupt_handlerR() - Funktion

```
void my_interrupt_handlerR()                    // Interrupt for Button Right
{
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();      // get current time in ticks from program start
  // If interrupts come faster than 200ms, assume it's a bounce and ignore, and a button can only trigger once per move
  if (turnMutex && (interrupt_time-last_interrupt_time>200))
  {
    turnMutex=0;                                 // locks movement for the current move
    ButtonR();
  }
  last_interrupt_time = interrupt_time;         // next trigger is 200 ms from now so last time is current time
}
```

Button() - Funktionen

```
void ButtonL(){
  direction--;
  if(direction <0){
    direction = 3;
  }
}
```

```
void ButtonR(){
  direction++;
  if(direction >3){
    direction = 0;
  }
}
```

```
// Interrupt buttons call these functions to change the direction

// directions  are defined as Right = 0, Down = 1, Left = 2 and Up = 3,
```

Aufgetretene Probleme

- Sinnvolle Verkabelung der Matrix
- Überlappung von Matrixelementen
- Vereinzelte Codeprobleme

Projektvorschau