LCD Hill Run – Ein Arduino Jump’n’Run

Wer den ganzen Tag Daten von Sensoren ausliest hat sich eine Runde Zocken am Abend verdient! Für dieses kleine Arduino Jump’n’Run benötigst du nur wenige Bauteile (und einige Kabel). Du baust es in wenigen Minuten auf und kannst danach gleich loslegen.

LCD Hill Run in Action

Die Idee und der Code für dieses Spiel stammt von Miles C., der es unter der Linzenz CC BY-SA 4.0 im Arduino Project Hub veröffentlicht hat. Wir erläutern nicht den ganzen Sketch des Spiels, aber erklären dir eine tolle Funktion, die du vielleicht noch nicht kennst: **attachInterrupt()**

Für dieses Projekt benötigst du:

- Arduino Uno*
- LCD-Display 16×2*
- Piezo-Summer*
- Potentiometer*
- 2 Buttons
- 1 Widerstand 220Ω
- 2 Widerstände 10kΩ

- Breadboard & Kabel

* *Amazon Affiliate Links – Wenn du dort bestellst, erhalten wir eine kleine Provision.*

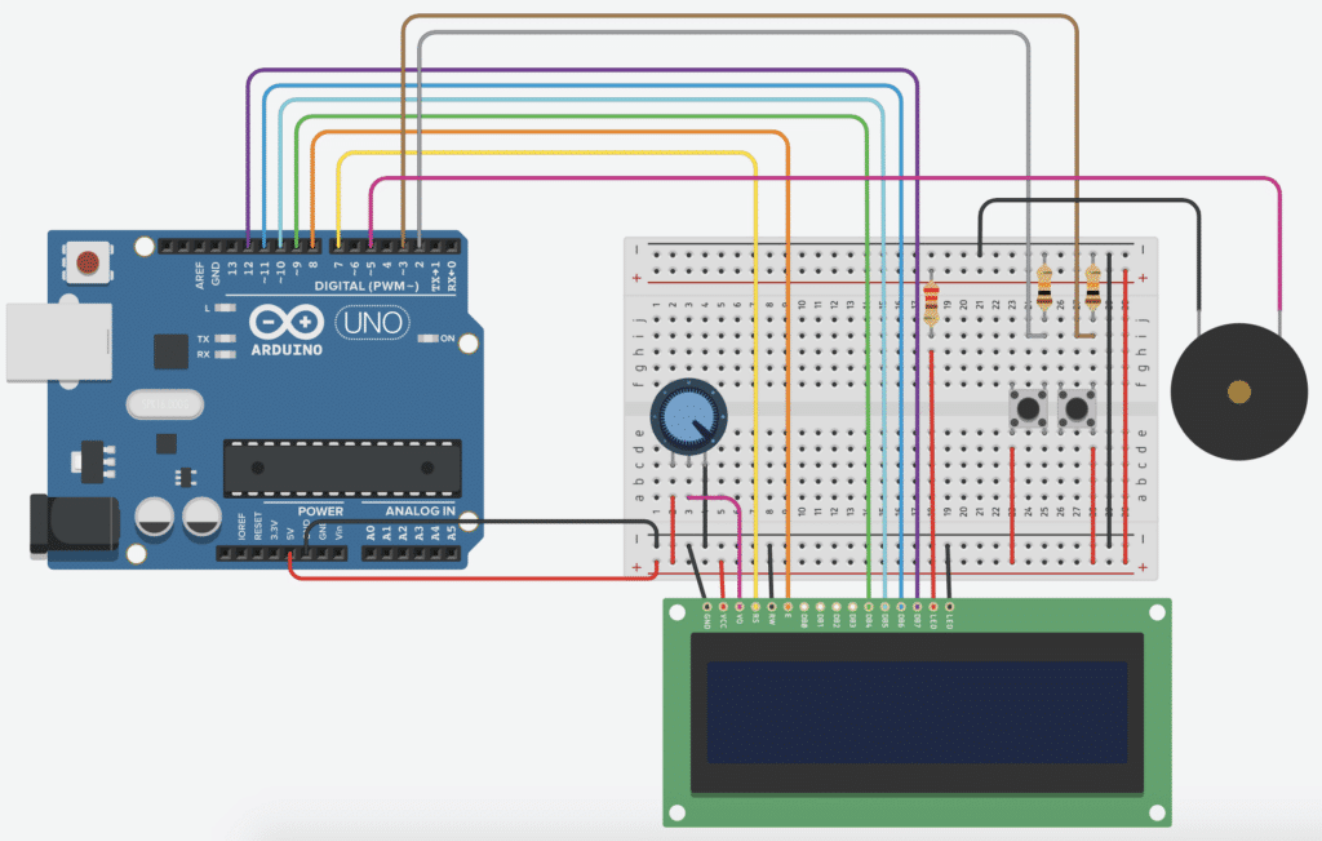
So funktioniert das Spiel

Die „Story“ des Spiels ist schnell erzählt: Du spielst ein kleine Figur, die auf einem LCD-Display von alleine rennt und Hindernissen ausweichen muss. Hierbei musst du entweder springen oder dich ducken. Für jedes überwundene Hindernis erhältst du einen Punkt. Sobald du an einem Hindernis hängen bleibst, hast du verloren und dir wird dein Punktestand angezeigt.

Der Aufbau des Projekts

Wie du auf dem Screenshot unten siehst, sind bei diesem Projekt viele Kabel im Spiel. Das LCD-Display benötigt besonders viele. Wenn du ein Display mit I^2C und integriertem Potentiometer verwendest, kannst du die Kabelei um einiges reduzieren.

Baue das Spiel wie folgt auf und prüfe alle Verbindungen noch einmal bevor du loslegst:



Screenshot: Tinkercad

Der Sketch des Arduino Jump’n’Runs

Wir werden an dieser Stelle nicht auf jedes Detail eingehen, viele der Konzepte im folgenden Sketch findest du in diesen Projekten und Tutorials auf pollux labs ausführlich erklärt:

- Ein LCD-Display am Arduino anschließen und verwenden
- Töne mit einem Piezo-Summer ausgeben
- Eigene Zeichen auf einem LCD-Display anzeigen

Eine Funktion im Sketch ist aber recht selten anzutreffen, obwohl sie sehr hilfreich sein kann: **attachInterrupt()**

Die Funktion attachInterrupt()

Schauen wir erst auf ein gängiges Konzept: In diesem Sketch möchtest du auf einen gedrückten Button reagieren.

```
void loop() {
  buttonState = digitalRead(buttonPin);
```

```
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

Hier wird zunächst mit **digitalRead()** der Zustand des Buttons gelesen und dann in einer if/else-Abfrage mit einer LED reagiert.

Das funktioniert, ist aber etwas unpraktisch, wenn du im Loop noch etwas vorhast als nur darauf zu warten, dass jemand den Button drückt. **Besser ist es da, das Überwachen des Buttons einen sogenannten Interrupt erledigen zu lassen, denn damit kannst du mehrere Prozesse gleichzeitig ablaufen lassen**. In diesem Spiel sind das das Auslesen der beiden Buttons und die Animationen auf deinem LCD-Display. So ein Interrupt kann folgendermaßen aussehen:

```
volatile int buttonState = 0;

void setup() {

  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
   attachInterrupt(digitalPinToInterrupt(buttonPin), pin_ISR,
CHANGE);
}

void loop() {
}

void pin_ISR() {
  buttonState = digitalRead(buttonPin);
  digitalWrite(ledPin, buttonState);
}
```

Wie du siehst, ist hier der Loop sogar leer, die Abfrage des Buttons befindet sich jetzt im Setup. Das bedeutet, du hast im Loop Platz für andere Dinge. Die Funktion **attachInterrupt()** hat folgende Syntax:

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

Hierbei spielen die Parameter **interrupt** (der zu überwachende Pin), **ISR** (die beim Drücken des Buttons ausgeführte Funktion) **und mode** (Wann wird getriggert?) eine Rolle. Ausführliche Informationen findest du in der Arduino-Referenz.

Zurück zum Sketch des Spiels: Sobald du einen der beiden Buttons drückst, wird eine entsprechende Funktion aufgerufen, die deine Figur entweder springen – **seeJumping()** – oder sich ducken – **seeDucking()** –lässt.

Ansonsten besteht der Großteil des Sketchs dieses Arduino Jump'n'Runs aus Abfragen und Animationen, die dich auf den ersten Blick vermutlich ziemlich überrumpeln. Aber mit etwas Übung und Zeit wirst du auch durchsteigen – wenn du das möchtest und nicht einfach nur spielen willst.

Hier der gesamte Sketch:

```
/*
 * Copyright (c) 2020 by Miles C.
*/

#include <LiquidCrystal.h>
#include "pitches.h"
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

const int JUMP_PIN = 2;
const int BUZZER_PIN = 5;
const int DUCK_PIN = 3;

const int JUMP_PITCH = 2700; //sounds when button pressed
const int JUMP_PITCH_DURATION = 50; //sounds when button pressed
```

```
const int DUCK_PITCH = 1350; //sounds when button pressed
const int DUCK_PITCH_DURATION = 50; //sounds when button pressed
const int DIE_PITCH = 200; //sounds on death
const int DIE_PITCH_DURATION = 500; //sounds on death
const int TICKSPEED = 90; //ms per gametick, 1 gametick per hill move.
const int JUMP_LENGTH = 3; //chars jumped over when jump is pressed.
const byte stickStep1[8] = {
  B01110,
  B01110,
  B00101,
  B11111,
  B10100,
  B00110,
  B11001,
  B00001,
};
const byte stickStep2[8] = {
  B01110,
  B01110,
  B00101,
  B11111,
  B10100,
  B00110,
  B01011,
  B01000,
};
const byte stickJump[8] = {
  B01110,
  B01110,
  B00100,
  B11111,
  B00100,
  B11111,
  B10001,
  B00000,
};
const byte stickDuck[8] = {
  B00000,
```

```
  B00000,
  B00000,
  B01110,
  B01110,
  B11111,
  B00100,
  B11111,
};
const byte hill[8] = {
  B00000,
  B00000,
  B01110,
  B01110,
  B01110,
  B11111,
  B11111,
  B11111,
};
const byte crow1[8] = {
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B01110,
  B01110,
  B01110,
};
const byte crow2[8] {
  B11111,
  B11111,
  B11111,
  B11111,
  B11111,
  B01110,
  B01110,
  B01110,
};

volatile int jumpPhase = JUMP_LENGTH + 1;
int gameTick = 0;
```

```
int crowX = 40;
int hillX = 25;
bool playerY = 0;
volatile bool ducking = LOW;
bool loopBreaker = 1;
bool crowGo = 0;
int score = 0;

void setup() {
  pinMode(JUMP_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  lcd.begin(16, 2);
  lcd.createChar(0, hill);
  lcd.createChar(1, stickStep1);
  lcd.createChar(2, stickStep2);
  lcd.createChar(3, stickJump);
  lcd.createChar(4, stickDuck);
  lcd.createChar(5, crow1);
  lcd.createChar(6, crow2);
  attachInterrupt(digitalPinToInterrupt(JUMP_PIN), seeJumping,
RISING);
  attachInterrupt(digitalPinToInterrupt(DUCK_PIN), seeDucking,
CHANGE);
}

void loop() {
  playerY = 0;
  if (jumpPhase < JUMP_LENGTH) {
    playerY = 1;
  }

  drawSprites();

  loopBreaker = 1;
  if (hillX < 16) {
    if (crowX < hillX) {
      hillX += 8;
      loopBreaker = 0;
    }
    if (loopBreaker) {
      lcd.setCursor(hillX, 1);
```

```
      lcd.write((byte)0);
    }
  }
  if (hillX < 1) {
    if (jumpPhase < JUMP_LENGTH) {
      score++;
      hillX = 16 + rand() % 8;
    } else {
      endGame();
    }
  }
  if (crowX < 16) {
    lcd.setCursor(crowX, 0);
    if (gameTick % 8 < 4) {
      lcd.write((byte)5);
    } else {
      lcd.write((byte)6);
    }
  }
  if (crowX < 1) {
    if (ducking) {
      score++;
      crowX = 24 + rand() % 16;
    } else {
      endGame();
    }
  }
  lcd.setCursor(0, playerY);
  lcd.print(" ");
  jumpPhase++;
  hillX--;
  crowGo = !crowGo;
  crowX -= crowGo;
  gameTick++;
  delay(TICKSPEED);
}

void endGame() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Score: ");
```

```
  lcd.setCursor(7, 0);
  lcd.print(score);
  tone(BUZZER_PIN, DIE_PITCH, DIE_PITCH_DURATION);
  while (!digitalRead(JUMP_PIN)) {
    lcd.setCursor(0, 1);
    if (millis() % 500 < 250) {
      lcd.print("Jump to Continue");
    } else {
      lcd.print("                ");
    }
  }
  lcd.clear();
  score = 0;
  hillX = 25;
  crowX = 40;
}

void drawSprites() {
  lcd.setCursor(0, 1 - playerY);

  if (!ducking) {
    if (!playerY) {
      if ((gameTick % 4) < 2 ) {
        lcd.write((byte)1);
      } else {
        lcd.write((byte)2);
      }
    } else {
      lcd.write((byte)3);
    }
  } else {
    lcd.write((byte)4);
  }
  lcd.setCursor(1, 1);
  lcd.print("               ");
  lcd.setCursor(1, 0);
  lcd.print("               ");
}
void seeJumping() {
  if (jumpPhase > (JUMP_LENGTH + 2) && !ducking) {
    jumpPhase = 0;
```

```
    tone(BUZZER_PIN, JUMP_PITCH, JUMP_PITCH_DURATION);
  }

}
void seeDucking() {
  ducking = digitalRead(DUCK_PIN);
  if (ducking) {
    jumpPhase = JUMP_LENGTH;
    tone(BUZZER_PIN, DUCK_PITCH, DUCK_PITCH_DURATION);
  }
}
```