

- Legende:**
- code in entsprechender Datei:
  - Funktion in entsprechender Datei:
  - Funktion von Datei außerhalb:
  - wenn eine Funktion von einer Datei von außerhalb aufgerufen wird, ist diese mit einem Pfeil verbunden

### LedMatrixInit.c

Liest Matrixnummer.c aus und schreibt die entsprechende Nummer um die Led anzusteuern an die passende Stelle.

Grund:  $matrix[0][0] = 883$   
 $matrix[30][30] = 0$

kein single ansteuern ohne diese Funktion aufgrund der Bauart nicht möglich

### Theme Song.c

**musical-score-encoder.c**  
Erzeugt aus Frequenz PWM-Signal für Buzzer

### DisplayInit.c

- **init\_i2c**
  - **Scan\_i2c\_devices**
- Esp-led & Lvgl Funktionen für Display ansteuerung

• Kommentare / Infos

### main.c

#### setup-led-strip

LedMatrixInit

esp-random = seed

Splash-show

led-strip clear & refresh

grid-init

score-init

score-load-highscore (from NVS)

show highscore on screen if available

init-controls

start Theme

start-game-loop

display-init

### grid.c

• **grid-init** (setzt Spielfeld komplett auf 0)

• **grid-check-collision** (schaut ob fallender Block irgendwo kollidiert)

• **grid-fix-block** (Speichert fallenden Block dauerhaft in Grid und rendert ihn auf Leds)

• **grid-clear-full-rows** (entfernt volle Reihen und lässt Blöcke darüber fallen und aktualisiert Score)

• **grid-print**

**grid-init**

**grid-check-collision**

**grid-fix-block**

**grid-clear-full-rows** → **score-add-lines**

**grid-print**

### Score.c

• **score-init**

• **score-add-lines**

1 →	100 pkt
2 →	300 pkt
3 →	500 pkt
4 →	800 pkt
>4 →	(lines + 900) pkt

• **score-load-highscore** (wird auf NVS gespeichert)

• **score-get-highscore**

• **score-update-highscore**

### GameLoop-NEW.c

• **spawn\_block**

+ **get-random-block**

+ **grid-check-collision**

+ **handle-game-over**

• **handle-game-over**

+ **led-strip-set-pixel**

+ **led-strip-refresh**

+ **display-show-game-over**

• **game-loop-test**

- **controls\_all\_buttons\_pressed()**
- **controls\_disable\_isr()**
- **controls\_enable\_isr()**
- **controls\_get\_event()**
- **controls\_wait\_event()**
- **splash\_show\_duration()**
- **splash\_show\_waiting()**
- **splash\_show()**
- **splash\_clear()**
- **grid\_init()**
- **score\_init()**
- **speed\_manager\_reset()**
- **speed\_manager\_init()**
- **speed\_manager\_update()**
- **speed\_manager\_get\_drop\_interval()**
- **display\_reset\_and\_show\_hud()**
- **display\_update\_score()**
- **score\_get\_highscore()**
- **grid\_check\_collision()**
- **grid\_fix\_block()**
- **grid\_clear\_full\_rows()**
- **get\_block\_rgb()**
- **Rotate\_block\_90()**
- **led\_strip\_set\_pixel()**
- **led\_strip\_clear()**
- **led\_strip\_refresh()**
- **spawn\_block()**
- **handle\_game\_over()**

### Splash.c

• **splash-show**

• **splash-clear** (wenn Spiel gestartet wird)

• **splash-show-duration** (Splash während Inputs ignoriert werden)

• **splash-show-waiting** (zeigt Splash screen bis Input, wird nach GameOver & Reset verwendet)

**splash-show**

Skalierte Tetris-Blöcke  
Laufschrift  
refresh only Text  
check Input  
loop

**splash-show-waiting**

Wie splash-show  
nur wird am Anfang  
Inputs ignoriert

**splash-show-duration**

Wie splash-show  
nur wird dies nach  
Game Over gezeigt

### Controls.c

**init-controls** Initialisiert GPIO-Pins, Event-Queue und ISR

**gpio\_isr\_handler** ISR-Handler, pusht Buttons in Queue

**check-button-pressed** Prüft ob Button gedrückt ist

**controls-get-event** Liest Button-Event aus Queue

**controls-wait-event** Wartet auf Button-Event aus Queue

**controls\_all\_buttons\_pressed** Prüft ob alle 4 Buttons gleichzeitig gedrückt sind (Reset)

**controls\_disable\_isr** Deaktiviert ISR für alle Buttons

**controls\_enable\_isr** Aktiviert ISR für alle Buttons

ISR = Interrupt Service Routine (Interrupt Handler)  
ermöglicht, dass Funktionen sofort ausgeführt werden

### Speed Manager.c

• **speed\_manager-init**

• **speed\_manager-get-fall-intervall**

• **speed\_manager-update-score**

• **speed\_manager-reset**

lines-cleared	speed (in ms/Led)
0	400
2	370
5	330
10	270
15	220
20	170
25	130
30	100
35	80
40	60
45	50

### Colors.c

gibt jedem Block die passende Farbe

- I - Cyan
- J - Blau
- L - Orange
- O - Gelb
- S - Grün
- Z - Rot
- T - Lila

### Blocks.c

• Form der Blöcke auf einer 4x4 Matrix

• **assign-block-color** (Weist jeden Block einen Farbindex zu)

• **get-block-rgb** (konvertiert Farbindex in rgb-Werte)

• **rotate-block-90**