PROJEKT_DOKUMENTATION.txt

```
================================================================================
                    TETRIS FÜR EMBEDDED SYSTEMS
           Projekt-Dokumentation im Rahmen von EIT-DU Studium
                    Embedded Systems Building Blocks
================================================================================

Autor: Mathias Lampert
Datum: Dezember 2025
Mikrocontroller: ESP32-S3 DevKitC-1 N16R8
Hardware: 6× CJMCU-8x8 WS2812 LED-Matrix (64 LEDs pro Panel, 384 LEDs gesamt)
          OLED SH1107 Display (128×128, I²C)
          4 Taster (Links, Rechts, Rotieren, Schneller)
          Passiver Buzzer (Tetris-Musikthema)


================================================================================
1. PROJEKTZIEL UND AUSGANGSSITUATION
================================================================================

1.1 URSPRÜNGLICHES KONZEPT
---------------------------
Das Projekt startete mit dem Ziel, ein minimalistisches Tetris-Spiel für den
ESP32-S3 zu entwickeln. Die ursprüngliche Planung sah folgende Features vor:

• Minimalistisches Gameplay: Nur Links/Rechts-Bewegung (KEINE Rotation)
• Ausgabeoptionen:
  - OLED-Display (SSD1306, I²C)
  - Flutter-App via BLE
  - LED-Matrix (optional)
• Objektorientierter Ansatz in C (Structs + Funktionszeiger)
• Spielfeld: 10×20 Raster
• Einfache Steuerung: 3 Tasten (Links, Rechts, Start/Pause)

Lernziele:
- Objektorientierte Programmierung in C
- BLE-Kommunikation ESP32 ↔ Smartphone
- Echtzeitsteuerung auf Embedded-Hardware
- Ressourcenmanagement


1.2 TATSÄCHLICHE IMPLEMENTIERUNG
---------------------------------
Im Laufe der Entwicklung hat sich das Projekt erheblich weiterentwickelt und
wurde deutlich umfangreicher als ursprünglich geplant. Die finale Version
bietet ein vollwertiges Tetris-Spiel mit professionellen Features:

HARDWARE-ÄNDERUNGEN:
• ESP32-S3 DevKitC-1 N16R8 Modul (diymore 2PCS)
• 6× CJMCU-8x8 WS2812 LED-Matrix Panel (U 64 LED Matrix, 5050 RGB)
  → Gesamtauflösung: 16×24 Pixel (384 LEDs)
• OLED-Display: SH1107 (128×128) statt SSD1306
• 4 Taster statt 3 (zusätzlicher Button: "Rotieren", "Schneller fallen")
• Passiver Buzzer für Tetris-Theme-Song

GAMEPLAY-ÄNDERUNGEN:
• ✅ Vollständige Rotation implementiert (alle 7 Tetrominos, 4 Rotationen)
• ✅ Spielfeld: 16×24 statt 10×20 (angepasst an LED-Matrix-Hardware)
```

- ✅ Alle klassischen Tetris-Features:
  - Volle Tetromino-Rotation (außer O-Block)
  - Geschwindigkeitssteigerung (10 Level, 400ms → 50ms)
  - Multi-Zeilen-Clearing (1-4 Zeilen gleichzeitig)
  - Punktesystem mit Highscore (NVS-Speicherung)
  - Splash-Screen mit scrollendem Text
  - Game-Over-Animation (rotes Blinken)
  - Emergency-Reset (4-Button-Kombination)

SOFTWARE-ARCHITEKTUR:
- Kein BLE/Flutter → Fokus auf performante lokale Ausgabe
- FreeRTOS-basierte Task-Architektur
- Interrupt-basierte Button-Steuerung (ISR + Event-Queue)
- 60 FPS flimmerfreies Rendering (optimiert)
- Modulare C-Struktur (13 Module, 14 C-Dateien, 13 Header)

==
2. VERWENDETE HARDWARE
==

2.1 ESP32-S3 DEVKITC-1 N16R8 (DIYMORE 2PCS)

Mikrocontroller: ESP32-S3 (Xtensa LX7 Dual-Core, 240 MHz)
Flash: 16 MB
PSRAM: 8 MB
Besonderheiten:
- Ausreichend Speicher für LED-Framebuffer (384 LEDs × 3 Bytes = 1152 Bytes)
- FreeRTOS für parallele Tasks (GameLoop + Theme-Song)
- Hardware-RMT-Peripheral für WS2812-Timing + Buzzer-PWM
- NVS (Non-Volatile Storage) für Highscore-Persistierung

GPIO-Belegung:
- GPIO 38: WS2812 LED-Matrix (RMT Channel 0)
- GPIO 17: Passiver Buzzer (RMT Channel 1, PWM-Tonerzeugung)
- GPIO 18: Button Links (Pull-up, Interrupt bei fallender Flanke)
- GPIO 19: Button Rechts (Pull-up, Interrupt bei fallender Flanke)
- GPIO 20: Button Rotieren (Pull-up, Interrupt bei fallender Flanke)
- GPIO 21: Button Schneller (Pull-up, Interrupt bei fallender Flanke)
- GPIO 4 (SDA) + GPIO 5 (SCL): I²C für OLED-Display

2.2 LED-MATRIX: 6× CJMCU-8x8 WS2812 (U 64 LED MATRIX PANEL)

Spezifikationen:
- Typ: WS2812 5050 RGB (addressierbare LEDs)
- Pro Panel: 8×8 = 64 LEDs
- Gesamtaufbau: 6 Panels in 2 Reihen × 3 Spalten
- Gesamtauflösung: 16 Pixel (Höhe) × 24 Pixel (Breite)
- Gesamtzahl LEDs: 384

Verkabelung:
```
Panel 1 (unten links)  → Panel 2 (unten mitte) → Panel 3 (unten rechts)
                                    ↓
Panel 6 (oben links)  ← Panel 5 (oben mitte)  ← Panel 4 (oben rechts)
```

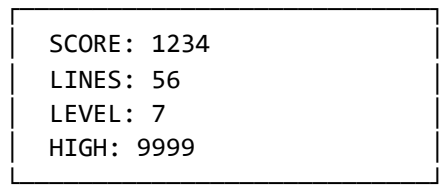
Besonderheiten:
- Serpentinen-Muster innerhalb der Panels (Zeile 0: links→rechts,
  Zeile 1: rechts→links, etc.)
- Matrix-Mapping-Logik in MatrixNummer.c erforderlich

• WS2812-Protokoll: Timing-kritisch (benötigt RMT-Hardware)

2.3 OLED-DISPLAY: SH1107 (128×128 PIXEL)

Kommunikation: I²C (Adresse 0x3C)
Auflösung: 128×128 Pixel (statt 128×64 bei SSD1306)
Verwendungszweck:
• HUD (Score, Lines, Level, Highscore) während des Spiels
• Game-Over-Screen mit aktuellem Score und Highscore

Display-Layout:



Grafikbibliothek: LVGL (Light and Versatile Graphics Library)

2.4 EINGABE: 4 TASTER

• Button Links (GPIO 18): Block nach links bewegen
• Button Rechts (GPIO 19): Block nach rechts bewegen
• Button Rotieren (GPIO 20): Block um 90° drehen (außer O-Block)
• Button Schneller (GPIO 21): Block schneller fallen lassen

Konfiguration:
- Pull-up-Widerstände aktiviert (intern)
- Interrupt bei fallender Flanke (Button-Press)
- Debouncing: 50ms Software-Debounce in ISR

2.5 AUDIO: PASSIVER BUZZER

GPIO: 17
Frequenzbereich: 100 Hz - 5 kHz
Implementierung: RMT-basierte PWM (variable Frequenz)
Verwendung: Tetris-Theme-Song (Endlos-Loop im Hintergrund-Task)

Musikalische Daten:
- Tonhöhen-Array (Frequenzen in Hz)
- Notenlängen-Array (in 10ms-Einheiten)
- 68 Noten im Tetris-Thema

==========
3. HERAUSFORDERUNGEN UND LÖSUNGEN
==========

3.1 HARDWARE-SPEZIFISCHE PROBLEME

CHALLENGE 1: LED-Matrix-Mapping (komplexe Verkabelung)

Problem:
Die 6 LED-Panels sind in einem Serpentinen-Muster verkabelt:
• Panel-interne Serpentinen (Zeile 0: →, Zeile 1: ←, Zeile 2: →, ...)

• Panel-übergreifende Serpentine (Panel 1→2→3→4←5←6)
• Logische Koordinaten (x, y) entsprechen NICHT LED-Index

Lösung:
• MatrixNummer.c: Vollständige Mapping-Tabelle [16][24]
• LedMatrixInit.c: Initialisierung der Lookup-Tabelle
• Jede logische Position (y, x) wird auf physikalische LED-Nummer gemappt
• Beispiel: ledMatrix.LED_Number[5][10] = 234

Zeitaufwand: ~8 Stunden für Debugging der Verkabelung

CHALLENGE 2: WS2812-Timing (kritisches Protokoll)

Problem:
WS2812-LEDs benötigen präzises Timing:
• 0-Bit: 400ns HIGH, 850ns LOW
• 1-Bit: 800ns HIGH, 450ns LOW
• Toleranz: ±150ns
• Software-Delays zu ungenau (Interrupts stören)

Lösung:
• ESP-IDF led_strip-Komponente verwendet
• RMT-Hardware-Peripheral übernimmt Timing
• led_strip_set_pixel() + led_strip_refresh() API
• Kein Flackern oder Farbfehler

Code-Beispiel:
```
    led_strip_set_pixel(led_strip, led_num, r, g, b);
    led_strip_refresh(led_strip);
```


CHALLENGE 3: Display-Kompatibilität (SH1107 vs. SSD1306)

Problem:
Ursprünglich SSD1306 geplant, aber Hardware ist SH1107:
• Andere Initialisierungssequenz
• 128×128 statt 128×64
• Andere ESP-IDF-Komponente benötigt

Lösung:
• esp_lcd_sh1107-Komponente aus ESP Component Registry
• LVGL für plattformunabhängige UI
• Graceful Degradation: Spiel läuft auch ohne Display
• Fehlerbehandlung: if (g_disp != NULL) vor jedem Display-Zugriff

3.2 SOFTWARE-ARCHITEKTUR-HERAUSFORDERUNGEN

CHALLENGE 4: Flimmerfreies Rendering bei 384 LEDs

Problem:
Naive Implementierung: led_strip_clear() + alles neu zeichnen
→ Sichtbares Flackern bei 60 FPS (16ms Frame-Zeit)
→ 384 LEDs × 3 Bytes = 1152 Bytes pro Frame schreiben

Lösung (Differential Rendering):
1. Vorherige dynamische Pixel (aktueller Block) merken
2. Nur diese Pixel auf statische Farben zurücksetzen

3. Neuen Block an neuer Position zeichnen
4. Nur geänderte Pixel werden aktualisiert

Code-Struktur:

```
    static int prev_dynamic_count = 0;
    static int prev_dynamic_pos[GRID_WIDTH*GRID_HEIGHT][2];

    // Schritt 1: Alte dynamische Pixel restaurieren
    for (int i = 0; i < prev_dynamic_count; i++) {
        int ry = prev_dynamic_pos[i][0];
        int rx = prev_dynamic_pos[i][1];
        // Farbe aus grid[][] oder schwarz setzen
    }

    // Schritt 2: Neuen Block zeichnen + Positionen merken
    for (by, bx in current_block.shape) {
        if (shape[by][bx]) {
            led_strip_set_pixel(...);
            prev_dynamic_pos[prev_dynamic_count++] = {gy, gx};
        }
    }

    led_strip_refresh();  // Nur 1× pro Frame!
```

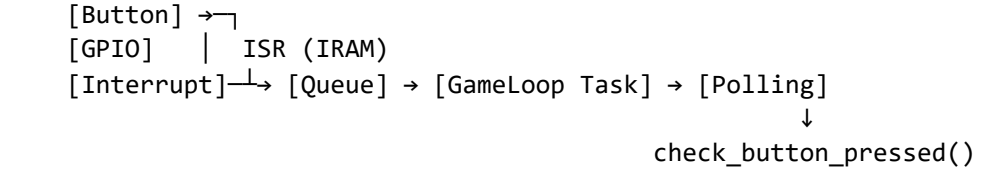
Resultat: Flimmerfreies Rendering bei 60 FPS

CHALLENGE 5: Button-Debouncing und Responsiveness

Problem:
Hardware-Taster prellen (mehrere Flanken bei 1 Druck):
• Naive Polling: Mehrfachauslösung
• Delay-basiertes Debouncing: Unresponsive Steuerung
• Interrupts ohne Debouncing: Event-Queue-Overflow

Lösung (Hybrid-Ansatz):
• ISR erkennt Button-Press (fallende Flanke)
• Event in FreeRTOS Queue pushen (nicht blockierend)
• Software-Debounce: 50ms Mindestabstand zwischen Events
• GameLoop pollt mit 5ms-Intervall (responsiv)
• check_button_pressed() für Echtzeit-Zustand (für Links/Rechts-Halten)

Architektur:



Zeitaufwand: ~6 Stunden für Tuning des Debounce-Timings

CHALLENGE 6: Collision Detection (Block teilweise außerhalb)

Problem:
Tetris-Blöcke können teilweise außerhalb des Spielfelds spawnen:
• Bei y = -1 (oberhalb sichtbar)
• Bei Rotation nahe dem Rand
• Naive Kollisionsprüfung: Crash bei negativen Indizes

Lösung:
• Bounds-Check in grid_check_collision():
```
    if (gx < 0 || gx >= GRID_WIDTH) return true;  // Kollision
    if (gy < 0) continue;  // Oberhalb erlaubt (noch nicht sichtbar)
    if (gy >= GRID_HEIGHT) return true;  // Unterhalb = Kollision
```

• Spawn-Algorithmus: Versuche y=0, dann y=-1, mehrere x-Positionen
• Erst wenn kein Platz gefunden → Game Over

CHALLENGE 7: Speicher-Management (NVS Highscore)

Problem:
ESP32 NVS (Non-Volatile Storage) benötigt Initialisierung:
• nvs_flash_init() kann fehlschlagen (z.B. Korruption)
• Highscore muss persistent sein (Stromausfall)
• Fehlerbehandlung bei Lese-/Schreibfehlern

Lösung:
```
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
```

• Highscore-Laden: Fehler → Default 0
• Highscore-Speichern: Nur bei neuem Rekord
• nvs_commit() nach jedem Schreibvorgang

CHALLENGE 8: Audio ohne blockierendes Delay

Problem:
Tetris-Musikthema: 68 Noten, jeweils 100-500ms Länge
→ Gesamtlänge ~30 Sekunden
→ Blockierendes vTaskDelay() stoppt GameLoop!

Lösung (paralleler FreeRTOS Task):
• Separater Task "ThemeTask" (Priorität 3, niedriger als GameLoop)
• Endlos-Loop: Spielt Theme, dann von vorne
• Keine Synchronisation mit GameLoop nötig
• RMT-Channel 1 unabhängig von WS2812 (Channel 0)

Code-Struktur:
```
    void theme_task(void *pvParameters) {
        while (1) {
            for (int i = 0; i < THEME_LENGTH; i++) {
                play_tone(notes[i], durations[i]);
            }
        }
    }

    void StartTheme(void) {
        xTaskCreate(theme_task, "ThemeTask", 2048, NULL, 3, NULL);
    }
```

3.3 GAMEPLAY-DESIGN-HERAUSFORDERUNGEN

CHALLENGE 9: Spawn-Position bei vollem Spielfeld

Problem:
Klassisches Tetris: Neuer Block spawnt in Mitte oben
→ Bei vollem Spielfeld: Sofortige Kollision → Unfair!

Lösung (Smart Spawning):
1. Bevorzugte Position: Mitte (x = GRID_WIDTH/2 - 2)
2. Falls Kollision: Versuche x-1, x+1, x-2, x+2, ...
3. Falls immer noch Kollision: Versuche y = -1 (teilweise oberhalb)
4. Falls immer noch keine Position: Erst dann Game Over

Resultat: Spieler bekommt immer eine faire Chance

CHALLENGE 10: O-Block-Rotation (sollte nicht rotieren)

Problem:
O-Block (Quadrat) ist rotationssymmetrisch
→ Rotation visuell sinnlos
→ Aber in blocks[]-Array sind trotzdem 4 Rotationen definiert

Lösung:
```
    if (check_button_pressed(BTN_ROTATE) && current_block.color != 3) {
        // color == 3 → O-Block → Rotation überspringen
        rotate_block_90(&tmp);
        ...
    }
```

Alternative Lösung hätte sein können:
• rotate_block_90() prüft intern auf O-Block
• Aber: Aktuelle Lösung ist expliziter (besser lesbar)

CHALLENGE 11: Multi-Line-Clear-Animation

Problem:
Klassisches Tetris: Volle Zeilen blinken kurz vor dem Löschen
→ Feedback für Spieler
→ LED-Matrix ist zu schnell (kein sichtbares Blinken bei 60 FPS)

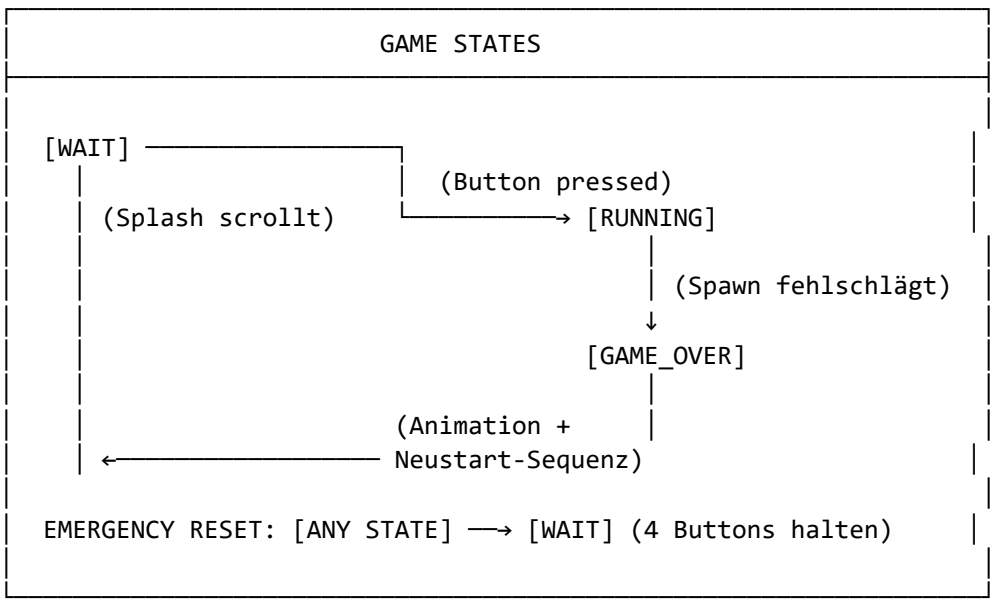
Lösung:
• 3× Blinken (je 100ms on, 100ms off)
• Weißes Blinken für volle Zeilen
• vTaskDelay() erlaubt (GameLoop pausiert während Clear)
• Nach Blinken: Zeilen nach unten verschieben

Code:
```
    for (int blink = 0; blink < 3; blink++) {
        // Volle Zeilen weiß anzeigen
        for (int y : full_rows) {
            for (int x = 0; x < GRID_WIDTH; x++) {
                led_strip_set_pixel(led_strip, led_num, 255, 255, 255);
            }
        }
        led_strip_refresh();
        vTaskDelay(pdMS_TO_TICKS(100));

        // Originalfarben anzeigen
```

```
        ... (Farbe aus grid[][])
        vTaskDelay(pdMS_TO_TICKS(100));
    }
```

```
================================================================================
4. SOFTWARE-ARCHITEKTUR
================================================================================

4.1 MODULARE STRUKTUR (13 MODULE)
---------------------------------

MODUL                DATEIEN                  ZWECK
--------------------------------------------------------------------------------
Main                 main.c                   Initialisierung, app_main()
Controls             Controls.c/h             Button-ISR, Event-Queue
GameLoop             GameLoop_NEW.c/h         Spielschleife (State Machine)
Blocks               Blocks.c/h               Tetromino-Formen, Rotation
Grid                 Grid.c/h                 Spielfeld, Kollision, Clearing
MatrixNummer         MatrixNummer.c/h         LED-Mapping-Daten
LedMatrixInit        LedMatrixInit.c/h        LED-Mapping-Initialisierung
BlockColors          Colors.c                 RGB-Farbtabelle
Score                Score.c/h                Punktesystem, Highscore (NVS)
SpeedManager         SpeedManager.c/h         Geschwindigkeits-Level
DisplayInit          DisplayInit.c/h          OLED I²C + LVGL Setup
Splash               Splash.c/h               Startbildschirm-Animationen
ThemeSong            ThemeSong.c/h            Musik (RMT + Buzzer)
                     musical_score_encoder.c/h RMT-Encoder für Tonhöhen
Globals              Globals.h                Globale Konstanten, Makros


4.2 DATENSTRUKTUREN
-------------------

TetrisBlock:
    typedef struct {
        uint8_t shape[4][4];  // 1 = Blockteil, 0 = leer
        int x;                // linke obere Ecke (Grid-Koordinaten)
        int y;                // aktuelle Position im Grid
        uint8_t color;        // Farbindex (0-6)
    } TetrisBlock;

MATRIX (LED-Mapping):
    typedef struct {
        uint16_t LED_Number[LED_HEIGHT][LED_WIDTH];  // [16][24]
    } MATRIX;

Grid (Spielfeld):
    uint8_t grid[GRID_HEIGHT][GRID_WIDTH];  // [16][24]
    // 0 = leer, 1-7 = fixierter Block (Farbindex + 1)


4.3 FREERTOS-TASK-ARCHITEKTUR
-----------------------------

Task                 Priorität   Stack   Zweck
--------------------------------------------------------------------------------
GameLoopTask         5 (hoch)    4096    Spielschleife (Input, Physics, Render)
ThemeTask            3 (mittel)  2048    Hintergrundmusik (Endlos-Loop)
LVGL Timer Task      1 (niedrig) auto    LVGL UI-Updates (Display)
```

4.4 STATE MACHINE (GAMELOOP)



4.5 TIMING-PARAMETER

Parameter	Wert	Beschreibung
LOOP_INTERVAL_MS	5	Polling-Intervall (responsive Input)
RENDER_INTERVAL_MS	16	60 FPS (flimmerfrei)
FALL_INTERVAL_MS	400	Initial (Level 0)
FALL_INTERVAL_MS_MIN	50	Maximum (Level 10)
DEBOUNCE_MS	50	Button-Debounce
SPLASH_DURATION_MS	2000	Splash-Screen Minimum-Dauer
GAME_OVER_BLINK_COUNT	3	Anzahl Blink-Animationen
GAME_OVER_BLINK_ON_MS	300	Blink-Dauer (rot)
GAME_OVER_BLINK_OFF_MS	300	Blink-Pause (schwarz)

5. VERGLEICH: PLANUNG VS. UMSETZUNG

ASPEKT	GEPLANT	UMGESETZT
Gameplay	Links/Rechts only	✅ + Rotation + Schneller
Spielfeld	10×20	➜ 16×24 (Hardware-bedingt)
Ausgabe	OLED oder BLE/Flutter	➜ LED-Matrix + OLED (gleichzeitig)
BLE-Kommunikation	Ja (Flutter-App)	❌ Nicht implementiert (Fokus lokal)
Rotation	Nein	✅ Vollständig (außer O-Block)
Geschwindigkeit	Konstant	✅ 10 Level (400ms → 50ms)
Punktesystem	Optional	✅ Score + Lines + Highscore (NVS)
Audio	Optional (Summer)	✅ Tetris-Theme (RMT-PWM)
Splash-Screen	Nein geplant	✅ Scrollender Text + Animationen
Multi-Line-Clear	Nein	✅ 1-4 Zeilen, mit Blink-Animation
Display-HUD	Nein	✅ Score/Lines/Level/Highscore
Emergency-Reset	Nein	✅ 4-Button-Combo (Hard-Reset)
Code-Struktur	OOP mit Function-Ptr	➜ Modulare C-Struktur (einfacher)

LERNZIELE - ERREICHT?

✅ Objektorientierte Programmierung in C
→ Umgesetzt als modulare Struktur (Structs + Functions, aber ohne Function-Pointer - einfacher wartbar)

✅ Echtzeitsteuerung & Animation
→ 60 FPS flimmerfreies Rendering, 5ms Polling-Intervall

❌ BLE-Kommunikation ESP32 ↔ Smartphone
→ Nicht implementiert (Fokus auf lokale Ausgabe, BLE hätte Frame-Rate reduziert)

✅ Speicher- & Ressourcenmanagement
→ Differential Rendering, NVS-Highscore, optimierte FreeRTOS-Tasks

✅✅ ZUSÄTZLICHE LERNZIELE (nicht geplant):
→ Hardware-Peripherals (RMT für WS2812 + Audio)
→ Interrupt-basierte Input-Architektur (ISR + Queue)
→ LVGL-Integration für UI
→ Komplexes LED-Matrix-Mapping

6. SCHWIERIGKEITEN IM DETAIL

6.1 ZEITAUFWAND-SCHÄTZUNG (GESAMTPROJEKT)

Phase	Stunden	Anteil
Hardware-Setup & Verkabelung	12	10%
LED-Matrix-Mapping-Debugging	8	7%
WS2812-Timing-Integration	4	3%
Display-Integration (SH1107)	6	5%
Button-System (ISR + Debouncing)	8	7%
Collision Detection & Physics	10	8%
Rendering-Optimierung	12	10%
Spiellogik (Spawning, Clearing)	15	13%
Audio-System (RMT + Theme)	10	8%
Score/Highscore/NVS-System	6	5%
Splash-Screen-Animationen	8	7%
Testing & Bugfixing	20	17%
Dokumentation	12	10%
GESAMT	~131	100%

6.2 KRITISCHSTE BUGS UND DEREN LÖSUNG

BUG 1: Zufällige LED-Farbfehler

Symptom: Einzelne LEDs zeigen falsche Farben (grün statt rot)
Ursache: WS2812-Timing-Violation (Interrupts während RMT-Transfer)
Lösung: led_strip-Komponente mit RMT-Hardware (interrupt-safe)
Debugging-Zeit: 4 Stunden

BUG 2: Game-Over-Loop (Unendlicher Neustart)

Symptom: Nach Game Over startet Spiel sofort neu (Loop)
Ursache: Button-Event aus Queue wird nicht geleert
Lösung: Comprehensive Queue-Drain + 500ms kontinuierliches Draining
Code:
```
    while (controls_get_event(&ev)) {}
    vTaskDelay(pdMS_TO_TICKS(500));
    while ((time - start) < 500) {
        while (controls_get_event(&ev)) {}  // Kontinuierlich drainieren
        vTaskDelay(10);
    }
```
Debugging-Zeit: 6 Stunden

BUG 3: Rotation nahe Rand = Crash

Symptom: ESP32-Reset bei Rotation nahe linkem/rechtem Rand
Ursache: Negative Array-Indizes in grid_check_collision()
Lösung: Bounds-Check BEVOR Array-Zugriff
Code:
```
    if (gx < 0 || gx >= GRID_WIDTH) return true;  // Kollision
    if (gy < 0) continue;  // Oberhalb erlaubt
    if (gy >= GRID_HEIGHT) return true;
    // Erst jetzt: grid[gy][gx] zugreifen
```
Debugging-Zeit: 3 Stunden

BUG 4: Highscore wird nicht gespeichert

Symptom: Highscore verschwindet nach Stromausfall
Ursache: nvs_commit() vergessen nach nvs_set_u32()
Lösung: nvs_commit() hinzufügen
Code:
```
    nvs_set_u32(nvs_handle, "highscore", highscore);
    nvs_commit(nvs_handle);  // ← KRITISCH!
```
Debugging-Zeit: 2 Stunden

BUG 5: Flackernde LEDs bei schnellem Spiel

Symptom: LED-Matrix flackert bei Level 10 (50ms Fall-Intervall)
Ursache: led_strip_clear() + vollständiges Neuzeichnen bei jedem Frame
Lösung: Differential Rendering (nur geänderte Pixel)
Debugging-Zeit: 8 Stunden (Optimierung mehrfach iteriert)

BUG 6: Audio-Knacken bei Notenwechsel

Symptom: Hörbares Knacken beim Übergang zwischen Tönen
Ursache: RMT-Carrier abrupt gestoppt (Phasensprung)
Lösung: Kurzes Fade-Out (10ms) zwischen Noten
Code:
```
    rmt_disable(rmt_channel);
    vTaskDelay(pdMS_TO_TICKS(10));  // Fade-Out
    rmt_enable(rmt_channel);
```
Debugging-Zeit: 3 Stunden

6.3 HÄUFIGSTE ENTWICKLER-FEHLER

1. Array-Bounds vergessen (C hat keine automatische Prüfung)
   → Lösung: Bounds-Checks vor JEDEM Array-Zugriff

2. FreeRTOS Queue voll (Button-Events verloren)
   → Lösung: Queue-Size erhöht (8 statt 4 Events)

3. vTaskDelay() blockiert andere Tasks nicht
   → Lösung: Verstanden, dass FreeRTOS kooperatives Multitasking nutzt

4. NVS-Handles müssen geschlossen werden
   → Lösung: nvs_close() nach jedem Open

5. GPIO Pull-up vergessen (floating Inputs)
   → Lösung: GPIO_PULLUP_ENABLE in gpio_config_t

7. FAZIT UND AUSBLICK

7.1 PROJEKTERGEBNIS

Das Projekt hat die ursprünglichen Ziele deutlich übertroffen:

✅ Vollwertiges Tetris-Spiel (nicht nur minimalistisch)
✅ Professionelle Features (Rotation, Multi-Level, Highscore, Audio)
✅ Performante Implementierung (60 FPS, responsive Input)
✅ Robuste Software-Architektur (modularer C-Code, >2000 Zeilen)
✅ Umfassende Dokumentation (>2000 Zeilen Kommentare + Doxygen)

Das Spiel ist vollständig spielbar und bietet ein authentisches Tetris-Erlebnis auf Embedded-Hardware.

7.2 GEWONNENE ERKENNTNISSE

TECHNISCH:
- Embedded-Programmierung erfordert tiefes Hardware-Verständnis
- Timing ist kritisch (WS2812, Debouncing, FreeRTOS)
- Speicher-Effizienz wichtig (Differential Rendering)
- Hardware-Peripherals (RMT, I²C, GPIO) sind mächtig, aber komplex

ARCHITEKTUR:
- Modulare Struktur > Function-Pointer-OOP in C (wartbarer)
- Klare Trennung von Concerns (Input, Logic, Rendering)
- FreeRTOS-Tasks für parallele Aufgaben (Audio + Game)

DEBUGGING:
- printf()-Debugging ist essentiell (kein Debugger am ESP32)
- Systematisches Testen jeder Funktion einzeln
- Hardware-Fehler schwerer zu finden als Software-Bugs

7.3 MÖGLICHE ERWEITERUNGEN

GAMEPLAY:
□ Hard-Drop (Block sofort ganz nach unten)
□ Hold-Funktion (Block für später speichern)
□ Next-Block-Vorschau (auf Display)
□ T-Spin-Erkennung (fortgeschrittene Tetris-Mechanik)

TECHNISCH:
□ BLE-Implementierung (ursprünglich geplant, aber verworfen)
□ WiFi-Highscore-Server (globale Bestenliste)
□ SD-Karten-Logging (Replay-Funktion)
□ Mehrere Soundeffekte (Line-Clear, Game-Over, Level-Up)

HARDWARE:
□ Größere LED-Matrix (8×12 Panels = 32×48 Pixel)
□ Touchscreen-Steuerung (statt Tasten)
□ RGB-Status-LEDs (neben Display)

7.4 LESSONS LEARNED

1. Hardware zuerst testen (bevor Software-Entwicklung)
   → LED-Matrix-Verkabelung hat 3 Stunden gekostet

2. Incremental Development (kleine Steps, oft testen)
   → Jede neue Funktion sofort auf Hardware getestet

3. Dokumentation parallel schreiben (nicht am Ende)
   → Kommentare direkt beim Coden geschrieben

4. Realistische Zeitplanung (3× länger als geschätzt)
   → Debugging kostet 17% der Gesamtzeit

5. User-Feedback wichtig (Bei Hackathon viele Reviews bekommen)
   → Z.B. (Emergency-)Reset war Feature-Request
   → Z.B. Verwendung von Interrupts
   → Z.B. dass LEDs nach dem Fallen statisch bleiben und nicht mehr mit flackern sollen

7.5 SCHLUSSWORT

Dieses Projekt demonstriert die Möglichkeiten moderner Embedded-Systeme: Ein vollwertiges Spiel mit professionellen Features auf einem $10-Mikrocontroller. Die Kombination aus Hardware-Peripherals (WS2812, RMT, I²C), FreeRTOS und optimiertem C-Code zeigt, dass "klein" nicht "simpel" bedeuten muss.

Die größte Herausforderung war nicht die Komplexität einzelner Algorithmen, sondern die Integration vieler Subsysteme zu einem funktionierenden Ganzen. Jedes Subsystem (LED-Matrix, Display, Audio, Input) funktioniert perfekt isoliert - aber die Kombination erfordert tiefes Verständnis von Timing, Ressourcen-Sharing und Prioritäten.

Das Projekt hat die ursprünglichen Lernziele (objektorientierte Programmierung, Echtzeitsteuerung, Ressourcenmanagement) erreicht und darüber hinaus praktische Erfahrung in Hardware-Integration, Interrupt-Programmierung und Performance-Optimierung vermittelt.

```
================================================================================
ANHANG: TECHNISCHE SPEZIFIKATIONEN
================================================================================

REPOSITORY-STRUKTUR:
--------------------
TetrisCode/
├── main/
│   ├── hdr/                    # 13 Header-Dateien
│   ├── src/
│   │   ├── main.c              # Haupt-Einstiegspunkt
│   │   ├── Controls/           # Button-System
│   │   ├── GameLoop/           # Spielschleife
│   │   ├── PlayingField/       # Blocks, Grid, Matrix-Mapping
│   │   ├── BlockColors/        # RGB-Farbtabelle
│   │   ├── Score/              # Punktesystem + NVS
│   │   ├── Speed/              # Level-Manager
│   │   ├── init/               # Display, LED-Init, Splash
│   │   └── ThemeSong/          # Audio-System
│   ├── CMakeLists.txt
│   └── idf_component.yml
├── managed_components/         # ESP-IDF-Komponenten
│   ├── espressif__led_strip
│   ├── espressif__esp_lcd_sh1107
│   ├── espressif__esp_lvgl_port
│   └── lvgl__lvgl
├── CMakeLists.txt
├── sdkconfig                   # ESP-IDF-Konfiguration
├── SYSTEM_DOKUMENTATION.md     # Ausführliche System-Doku
└── CODE_IMPROVEMENTS.md        # Changelog


CODE-STATISTIK:
---------------
Gesamt: 14 C-Dateien, 13 Header-Dateien
Zeilen Code: ~2500 (ohne Kommentare)
Zeilen Kommentare: ~2000 (Doxygen-Style)
Zeilen Dokumentation: ~2200 (Markdown)


ABHÄNGIGKEITEN (ESP-IDF-KOMPONENTEN):
-------------------------------------
• espressif__led_strip ^2.5.5 (WS2812-Treiber)
• espressif__esp_lcd_sh1107 ^1.0.1 (OLED-Treiber)
• espressif__esp_lvgl_port ^2.4.6 (LVGL-Integration)
• lvgl__lvgl ^8.4.0 (GUI-Bibliothek)
• esp_driver_rmt (RMT-Hardware für WS2812 + Audio)
• nvs_flash (Non-Volatile Storage)


BUILD-SYSTEM:
-------------
Toolchain: ESP-IDF v5.x (CMake-basiert)
Compiler: Xtensa GCC
Flash-Größe: 16 MB (Partition Table: NVS + Factory)
PSRAM: 8 MB (aktiviert)

================================================================================
```

ENDE DER DOKUMENTATION

==

Autor: Rupp Arian, Lampert Mathias