



OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Projektbericht

Datenverarbeitung in der Technik

Gruppe 5 - Arcade LED

Sommersemester 2023

Mitglieder:

Lisa-Marie Dengler MatNr.: 3214589

Lukas Landgraf MatNr.: 3269224

Khan Ali Muttaqy MatNr.: 3289316

Reilly Ertman MatNr.: 3283484

Inhaltsverzeichnis

1 Einleitung	2
1.1 Ausgangssituation	2
1.2 Ziele und Erwartungen	2
2 Allgemeine Informationen	3
2.1 Teamvorstellung	3
2.2 Aufgabenbereiche und Arbeitsteilung	3
2.3 Technische Spezifikationen	4
3 Vorüberlegungen und Entwurfsphase	5
3.1 Vorüberlegungen (Reilly)	5
3.2 Vorüberlegungen	5
3.3 Konzept (Reilly)	5
3.4 Konzept	6
4 Design und Architektur (Lukas)	6
5 Implementierung	7
5.1 LED-Ansteuerung und Hardware (Reilly)	7
5.1.1 WS2812B LED Protokol	7
5.1.2 Bitübertragungsschicht des ESP32 und das RMT Peripheral	9
5.1.3 LED Benutzeroberfläche	10
5.1.4 Kritischer Bereich und Double Buffering	10
5.1.5 LED Steuerungsfunktionen und rmtWriteItems (Khan, du kannst diese section nennen, wie du magst) and LED Memory Management	11
5.1.6 Flackernde Frames (Lukas)	11
5.2 Backend (Lukas)	12
5.2.1 Netzwerk	12
5.2.2 Webserver	13
5.2.3 Spielesteuerung	14
5.2.4 Spieldaten und Taskkommunikation	15
5.3 Frontend (Lisa-Marie)	17
5.3.1 Vorüberlegungen	17
5.3.2 Struktur und Aufbau	18
5.3.3 Seitenspezifikationen	20
5.3.4 Design und Kommunikation	24
5.3.5 Einbindung neuer Spiele	26
5.3.6 Idle-LED-Anzeige	26
5.4 Spiele	27
5.4.1 Othello/ReversiXT (Lukas)	27
5.4.2 Snake (Lisa-Marie)	32
6 Aufwandsdokumentation	35
6.1 Reilly Ertman	35

6.2 Khan Ali Muttaqy	35
6.3 Lukas Landgraf	35
6.4 Lisa-Marie Dengler	35
7 Fazit	35
8 Quellen	36
Abbildungsverzeichnis	36

1 Einleitung

Das Modul "Datenverarbeitung in der Technik", dient dem Zweck im Rahmen der Vertiefungsmodule des 3. Studienabschnittes, den Studierenden das Zusammenspiel von Hard- und Software in einer komplexen Aufgabenstellung zu vermitteln. Dabei wird eigenständig in selbst organisierten Gruppen eine Projektidee umgesetzt. Angefangen von der Planung bis hin zur Testung, werden Inhalte praktisch vermittelt und Wissen aus vorangehenden Semestern kombiniert. Alle Bereiche der Produktentwicklung werden durchlaufen und wertvolle Erfahrung kann in produktiver Teamarbeit gewonnen werden. Das Ende des Kurses beinhaltet eine Vorführung und Präsentation des abgeschlossenen Projektes.

1.1 Ausgangssituation

Unser Projekt wird die Überarbeitung eines der Vorgängerprojekte beinhalten. Es handelt sich dabei um eine Sperrholzplatte, worauf mit LED-Bändern ein quadratisches Display aufgeklebt wurde. Insgesamt handelt es sich um 30x30 LED's welche alle separat ansteuerbar sind. Zur Verbesserung der Sichtbarkeit und des Schutzes der Hardwarekomponenten ist eine Plexiglasscheibe mit Milchglasfolie angebracht.

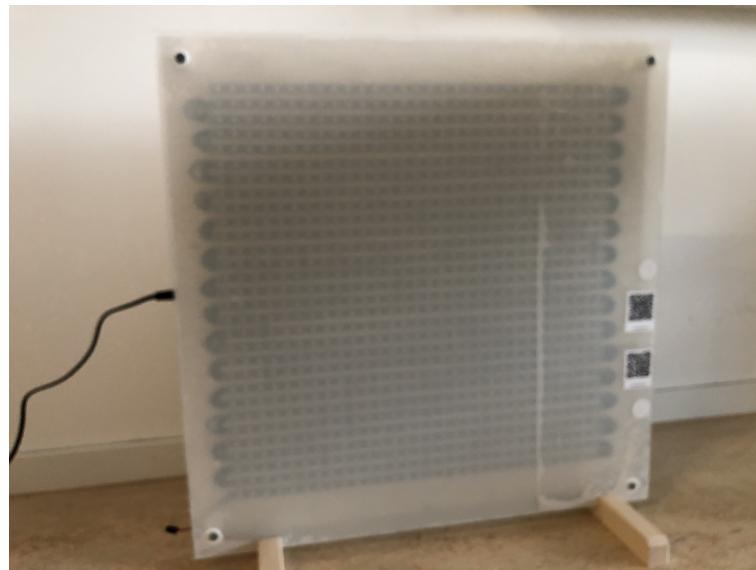


Abbildung 1: Foto des Boards

1.2 Ziele und Erwartungen

Das Vorgängerteam hat eine gute Arbeit geleistet. Unsere Aufgabe ist es, die Schwächen des Vorgängerprojektes auszugleichen und eigene Verbesserungen/Ideen zu integrieren. Unser Augenmerk liegt auf einer gut ausgearbeiteten, stabilen und sicheren Verbindung vom Endgerät mit dem LED-Board. Das Webinterface muss benutzerfreundlich, intuitiv und unabhängig vom Betriebssystem des jeweiligen Gerätes nutzbar sein. Es ergeben sich folgende Ziele:

- Auf dem LED-Board sollen grafisch reduzierte Spiele gespielt werden können, wie zum Beispiel „Snake“, „Tetris“ oder „Pong“.
- Die Hardware wird mit einem Microcontroller gesteuert, welcher zusätzlich ein lokales Netzwerk aufsetzt.
- Spieler sollen in der Lage sein, sich in dieses Netzwerk mit ihren privaten Mobilgeräten zu verbinden.
- Vom Microcontroller wird eine Webanwendung gehostet, welche als Steuerungseinheit dient.
- Von der Webanwendung aus können gewählte Spiele gestartet werden und auch ein Mehrspielermodus wird verfügbar sein.

Hinzu kommen folgende Erwartungen an das Modul:

- Vertiefen des gesamten Soft-/Hardware Entstehungsprozesses
- On-Hand Erfahrung im Team
- Vertiefung des Verständnisses von Embeddedprojekten
- Vertiefung der Programmierkenntnisse

Da die Projektaufgabe im teamorientierten Ansatz gelöst werden soll, können hier Soft Skills trainiert und erprobt werden, die zur Koordination von Arbeitspaketen hilfreich und notwendig sind, wie zum Beispiel Teamfähigkeit, Kommunikationsfähigkeit, Selbstorganisation, Projektorganisation, Projektmanagement, Verlässlichkeit und Kritikfähigkeit.

2 Allgemeine Informationen

In diesem Kapitel soll ein Einblick in die Teamstruktur und die Voraussetzungen, auf deren Basis das Projekt entwickelt wurde, gegeben werden. Die Beschreibung der technischen Hilfsmittel und verwendeten Tools soll einen tieferen Einblick in die Arbeitsweise unserer Gruppe vermitteln.

2.1 Teamvorstellung

Unsere Gruppe besteht aus vier Teammitgliedern, Lukas Landgraf, Reilly Ertman, Khan Ali Muttaqy und Lisa-Marie Dengler. Wir befinden uns alle im sechsten Semester der Technische Informatik (IT6). Alle Mitglieder der Gruppe verfügen über Kenntnisse der vorangehenden Semester und eigens vertiefte praktische Fähigkeiten, welche insgesamt das Team gut vervollständigen.

2.2 Aufgabenbereiche und Arbeitsteilung

Für verbesserte Effektivität und Zusammenarbeit an diesem Projekt, haben wir uns dazu entschieden, die Umsetzung in Aufgabenbereiche aufzuteilen.

- Reilly Hardwareansteuerung, LEDs und Hardware/Software Schnittstelle
- Khan - Hardwareansteuerung und LEDs
- Lukas - Backend, Schnittstellenkommunikation und ReversiXT
- Lisa-Marie - Frontend, Snake und Gruppenorganisation

2.3 Technische Spezifikationen

In diesem Projekt werden die Programmiersprachen C/C++ unter dem Standard C++20 verwendet. Der Grund dafür liegt im Erfahrungswert und Kenntnisstand bezüglich dieser Programmiersprache, da alle Gruppenmitglieder mit am besten vertraut sind.

C++20 ist auch der aktuellste Standard, der zur Verfügung steht. Dadurch wird ein größtmöglicher Umfang an Funktionalitäten durch Programmbibliotheken bereitgestellt.

Das Hostbetriebssystem aller verwendeten Rechner ist Windows 10/11, 64 Bit.

Die verwendete IDE ist *Visual Studio Code* (V 1.78.0 user setup).

Bei dem verwendeten Microkontroller handelt es sich um einen *ESP32 Wroom-32u*. Dieser ist in der Lage ein eigenes Netzwerk zu hosten und verfügt über ausreichend Speicherplatz für die Webseite, Netzwerkhosting, die Schnittstellen und die Logik der Spiele.

Um die Umsetzung des Projektes möglichst gut zu bewerkstelligen, verwenden wir die folgenden Tools:

- Zum Kompilieren der Quellcodes benutzen wir *Cmake* Version: 3.16
- In Visual Studio Code verwenden wir die *PlatformIO* Erweiterung
- Abgestimmt auf PlatformIO verwenden wir das Framework *ESP-IDF*.
- Das Frontend der Webanwendung verwendet HTML, CSS und Javascript.
- Verschiedene Bibliotheken zum Ansteuern der LED's.
- Für Versionskontrolle und gemeinsames Arbeiten nutzen wir *GitLab*
- Zur Kommunikation innerhalb der Gruppe verwenden wir *Discord*.
- Erstellung des Projektberichts mit *TEXMaker* Version: 5.1.3

3 Vorüberlegungen und Entwurfsphase

3.1 Vorüberlegungen (Reilly)

Wir setzen das Projekt von Studierenden aus letztem Semester fort. Um zu spielen musste man ein App herunterladen und dies verwenden als Spielcontroller. Dann verbindet man das Handy mit dem Mikrocontroller über Bluetooth. Das Spielen über Bluetooth hat sich als unstabil und unzuverlässig bewiesen. Immer wieder war die Verbindung weg. Auch die Ansteuerung über ein App ist nicht ideal, denn Apps sind dadurch HandyHersteller- und Handyversion abhängig. Das App aus der Gruppe vom letzten Semester muss immer wieder überprüft und ggf. gewartet werden, wenn eine neue Handyversion veröffentlicht wird.

Deswegen haben wir ein Mikrokontroller aufgesucht, das sowohl ein eigenes WLAN Netzwerk als auch eine Webseite hostet. Spielansteuerung über eine Webseite ist Handyhersteller- und Handyversion unabhängig, was die Wartungskosten deutlich sinkt. Um zu spielen muss man nur zwei QR Codes scannen um loszulegen, eines um zur WLAN zu verbinden und ein zweites um die Webseite aufzurufen.

Ein integriertes WLAN und gehostete Webseite die vor allem gleichzeitig laufen bedeutet viel Rechenlast. Wenn parallel ein Spiel und noch 900 LEDs gesteuert werden soll, muss das gewählte Mikrokontroller zwingend merhkernig sein.

Da die 900 ws2812b LEDs asynchron und seriell verbunden sind, waren die Auswahl an Kommunikationsprotokolle eingeschränkt. Die LEDs besitzen keinen Ausgang um ein ACK zu senden, also ist I2C nicht möglich. Außerdem haben die LEDs spezielle Timing-Anforderungen (mehr dazu im Kapitel Implementierung). Um Bit Information als Farbe und Helligkeit zu dekodieren müssen die LEDs ständig und immer in der gleichen Geschwindigkeit Information vom Mikrokontroller fangen. Als Protokoll eignet sich UART auch nicht: Es gibt keine Baudrate mit einer Fehlerquote unter 1 Prozent in der Geschwindigkeit, die die LEDs brauchen um Bit Information in Farbe und Helligkeit zu dekodieren. Es bleibt also wenige asynchrone, serielle Protokolle übrig. Am besten wäre ein eingebettetes Waveform-Generator, womit wir direkt mit den LEDs kommunizieren könnten. Dafür bietet das ESP32 das RMT Peripheral an (mehr dazu in Implementierung).

3.2 Vorüberlegungen

3.3 Konzept (Reilly)

Unser Ziel war es die Zeit und der Aufwand ein Spiel zu starten und zu zocken möglichst kurz zu halten, sonst ist die Motivation zum Zocken weg bevor das Spiel begonnen hat. Man steckt das Board an, verbindet sich mit dem Netzwerk, ruft die Webseite auf, wählt das Spiel aus dem Menü aus und Voilà! Schon ist man am Zocken.

3.4 Konzept

4 Design und Architektur (Lukas)

Das Softwaredesign soll es ermöglichen, Spiele ohne große Verzögerung zu starten, zu spielen und zu beenden, sodass das ganze Gerät nicht neugestartet werden muss. Gleichzeitig können Spiele nachträglich hinzugefügt werden, ohne etwas an der Implementierung der LED Steuerung oder dem Webserver ändern zu müssen. Es muss zudem möglich sein, dass alle Teammitglieder möglichst parallel arbeiten können und nicht auf andere warten müssen.

Aus diesem Grund kann unser Design in vier Blöcke unterteilt werden (siehe Abbildung 2):

- das Backend, welches das Netzwerk, den Webserver und das Steuern von Spielen implementiert
- die LED Board Steuerung, die eine simple Schnittstelle für den Spieleentwickler liefern soll, um das Board ohne Schwierigkeiten steuern zu können
- die Website, welche mit dem Backend für die Spielesteuerung kommuniziert
- die Spiele selber, die all diese Schnittstellen nutzen

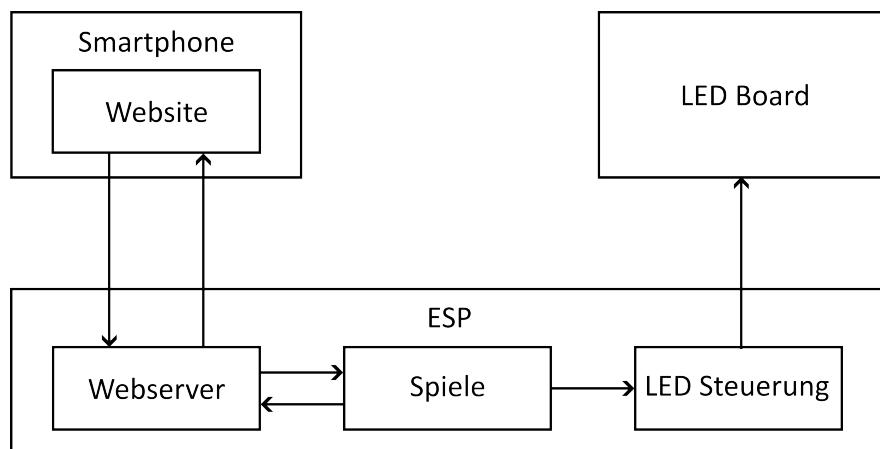


Abbildung 2: Darstellung der unterschiedlichen Blöcke und deren Schnittstelleninteraktion in unserer Implementierung.

Da diese Blöcke parallel arbeiten müssen, um schnelle Spielsteuerung zu gewährleisten, wird hierfür ein Real Time Operating System benötigt, welches es ermöglicht, die einzelnen Aufgaben auf Tasks aufzuteilen. Hierfür haben wir uns für das ESP-IDF entschieden, das ein Framework für die Entwicklung auf ESP Mikrokontrollern ist [1]. Mithilfe von FreeRTOS, welches in ESP-IDF enthalten ist [1], können die einzelnen Blöcke, wie Webserver und Spiele, auf eigenen Tasks laufen. Zusätzlich bietet ESP-IDF auch eine Menge nützlicher APIs für z.B. Webserver oder die Bedienung der Pins an [1].

5 Implementierung

5.1 LED-Ansteuerung und Hardware (Reilly)

Am GPIO des ESP32 muss eine Spannung erzeugt werden. Diese elektrische Spannung fließt dann durch eine Leitung an den Eingang des ersten asynchronen ws2812b LED in der Kette. Die Spannungen müssen bis zu einer Granularität von 50ns einstellbar sein. Die Erzeugung der Spannung soll so rechnenschonend sein wie möglich. Die LEDs müssen dieses Protokoll verstehen.

Hierfür biete das ESP32 das RMT (Remote Control Transceiver) Peripheral an. Da dieses eingebettete Waveform-Generator Signale alle 10ns erzeugt und Direct Memory Access (DMA) benutzt, was der Last an das CPU deutlich verringert, haben wir uns für das RMT Peripheral entschieden.

5.1.1 WS2812B LED Protokol

Ein ws2812b LED besitzt folgedes Interface: ein einziges Data_in, Masse und VCC. Pro LED werden 24 bit Information benötigt um das LED zu beleuchten. Die 3 byte eines LED Array besteht aus 3 Teilen: Ein byte für das Grüne, das Rote und das Blaue Anteil jeweils. Umso höher die jeweilige Byte-Werte, desto heller leuchtet das LED. Um das LED Grün zu beleuchten, setzt man das erste Byte auf 0, das zweite auf 255 und das letzte Byte auf 0. Um Weiß auszugeben, lauten die Byte-Werte 255/255/255.

Da wir nur ein einzelnes Data_in am Interface zur Verfügung haben, ist der Data_out vom ersten LED das Data_in vom zweiten LED, usw. Die ws2812B Kette bildet also ein Daisy Chain. Um das letzte von 900 LEDs in der Kette anzuleuchten, müssen diese Daten erst durch alle 899 LEDs durchgeschoben werden. Genau müssen diese 21600 Bitstellen (3 byte * 900 * 8) passieren. Dieses Verfahren heißt "Bit Banging". Man stelle sich vor jedes ws2812b wäre ein Schieberegister mit 24 bit jeweils. Wenn das erste Bit in der Leitung das Least Significant Bit des letzten LEDs in der Kette erreicht, sind die Daten dann an der richtigen Stelle.

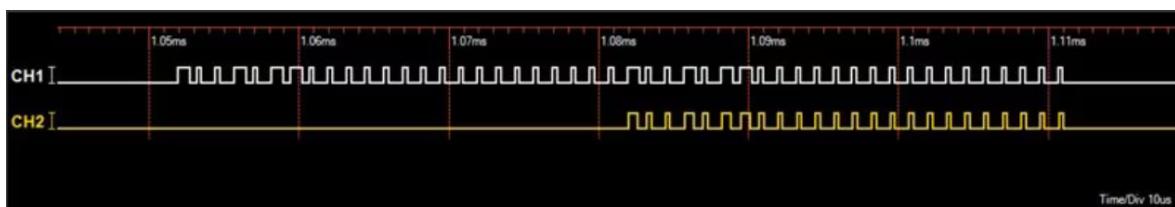


Abbildung 3: WS2812b liest eingehend Spannungen bzw. Wörter an der Leitung.

Abbildung 3 zeigt eine Oscilloscope Ausgabe eines Datenpakets. CH1 beinhaltet 48 bit binäre Information am Eingang des ersten LEDs und CH2 ist der Signal am Ausgang gemessen. Beide Signale beinhalten mehrere steigende und fallende Flanken. Die dargestellten analogen Signale sind vom Mikrocontroller erzeugt mit spezifischen Timing Anforderungen; nämlich, die Kombination aus einer steigenden und fallenden Flanke bildet ein Wort. Dieses vom Mikrokontroller enkodiertes analoges Signal wird dann

vom WS2812b als Wort dekodiert und bildet entweder ein Logical 1 oder 0. Kommt 24 solche Signale, können wir ein einzelnes LED anleuchten.

In dem Beispiel aus Abbildung 3 werden eine Reihe an Spannungen an das WS2812b geschickt. Konkret wird das folgende Wort zwei mal hintereinander dekodiert: 10010110000000000000000000000000. Die ersten 8 bit "10010110" beinhalten die Helligkeit für das Grüneanteil des LEDs. Rot und Blau haben den Wert 0. Weil das Wort zwei mal hintereinander steht, werden am Ende 2 LEDs Grün leuchten. Danach wird die Leitung auf 0 gezogen. Wenn 0 konstant nach einer Übertragung ausgegeben wird, kommt keine weiter Information und die LEDs sollen ihren letzten Zustand behalten.

WS2812B Timings: Jedes Bit ist also vom ESP32 enkodiert als ein Pulsweltenmodulationssignal mit einem gewissen Duty Cycle. Das Datenblatt vom Hersteller erläutert genauer über die Timing der Pulsweltenmodulation in Abbildung 3 and 4.

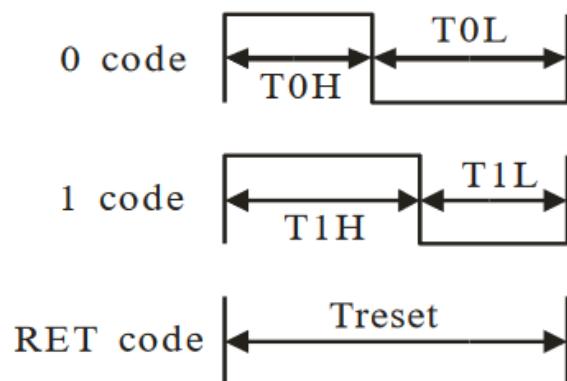


Abbildung 4: Ein ws2812b Wort besteht aus einem high gefolgt von einem low Part.

Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Abbildung 5: WS2812b Die Länge eines Wortes in Nanosekunden.

Sollte ein LED Grün leuchten, müssen die ersten 8 von 24 bits auf '1' gesetzt sein und der Rest auf '0'. Für jedes gesetztes Bit muss am Output Pin des ESP32 ein "high"SSignal für 0,8us gefolgt von einem "low"SSignal für 0,45us ausgegeben werden. Die Kombination aus T1H und T1L versteht ein WS2812B LED als ein Logical 1. Soll ein Bit nicht gesetzt werden, erzeugt das ESP32 am Output Pin ein "high"Pegel für 0,4us gefolgt von einem "low"Pegel für 0,85us. Daraus versteht das ws2812b das Signal als ein logical 0. Wenn für mindestens 50us lang am Dataline 0V Signal anlegt, behalten die LEDs ihren Zustand.

5.1.2 Bitübertragunsschicht des ESP32 und das RMT Peripheral

Wenn man Daten an ein WS2812b LED senden möchte, gibt es ein Problem. Das WS2812b Protokol ist asynchron und bietet keine Möglichkeit von einem Hardware Handshake gebrauch zu machen. Das heißt wir müssen den größten gemeinsamen Teiler von T0H, T1H, T0L und T1L nehmen als Taktfrequenz, also 50 ns. Das ist nun die Granularität, mit dem wir die benötigten steigenden und fallenden Flanken bilden können, um mit dem Ws2812b kommunizieren zu können. Um Signale alle 50ns auszugeben, bietet das Esp32 einen eingebetteten und schlichten Waveform Generator an: das Remote Control Transceiver, genannt als RMT.

Das ESP32RMTChannel.h Headfile kümmert sich um die Erstellung des RMT-Modules als Transceiver. Die Funktion ConfigureForWS2812x() instanziert das RMT am gewünschten GPIO und mit gewünschten Prescaler. Da das Clock der RMT bei 80MHz liegt, wählen wir einen Prescaler von 4 aus um eine Taktfrequenz von 20MHz bzw. 50ns zu erreichen.

Das ESP32 biete zwei legacy libraries, die wir zur Instanziierung und Verwendung des RMTs benötigen: rmt_config_t und rmt_types_legacy. rmt_legacy.c biete die Funktion rmt_write_items() und erzeugt die gewünschten getaktete Spannung am GPIO. Wie bereits besprochen passieren auf dem data_in des LED Boards eine Reihe von steigenden und fallenden Flanken, wohingegen der Programmierer ist dafür zuständig die gewünschten Bits in den Buffer zu schaffen, die das LED Board anleuchten soll. Aber was passiert dazwischen und wie werden die digitale 1s and 0s auf dem Bildschirm zu getakteten Spannung am GPIO übersetzt? Dies erledigt der Datatype rmt_item32_t aus rmt_types_legacy.h für uns.

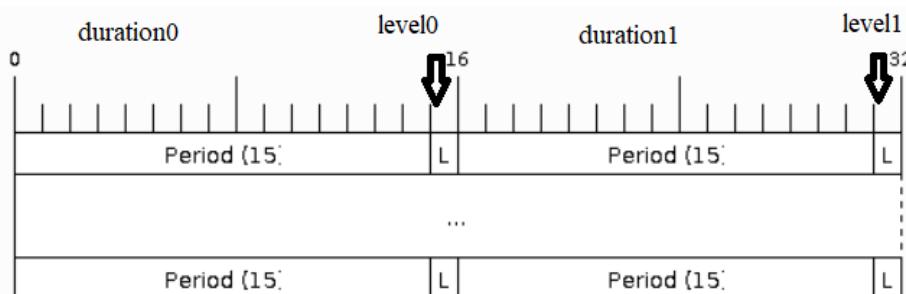


Abbildung 6: WS2812b reads timed high and low voltages and 1s and 0s.

rmt_item32_t ist ein 32-Bit integer union struct bestehend aus 4 Elementen in chronologischer Reihenfolge: duration0, level0, duration1, level1. Ein Union Struct heißt alle Mitglieder teilen den gemeinsamen Datenbereich. Die Bitbelegung der Mitglieder laut Abbildung 5 sieht aus wie gefolgt:

- rmt_item32_t(31 downto 17) = duration0
- rmt_item32_t(16) = level0
- rmt_item32_t(15 downto 1) = duration1
- rmt_item32_t(0) = level1

Um ein T0H auszugeben, setzen wir Duration0 auf den in DLEDController schon ausgerechneten Wert und level0 auf high. Um T0L auszugeben, setzen wir Duration1 auf den entsprechenden Wert und dann level1 auf 0. T0H + T0L wird am Esp32 GPIO dann als ein Logical 0 vom ws2812b verstanden.

5.1.3 LED Benutzeroberfläche

Ein LED Brett wird als Objekt der Klasse DStripData instanziert. Das Brett-Objekt besitzt ein 3D Array von $3 * \text{NUM_OF_LEDS}$ uint8 _t. Wenn der C/C++ Compiler ein x-D Array in Speicher kompiliert, wird Speicher mit einem Uni-Dimensionales Array beschrieben. Daten des kleinsten Indizes werden zusammenhängend geschrieben, wie gefolgt:

```
Array [2] [5] [3] = Array_1[5] [3], Array_2[5] [3];
Array_1[5] [3]={ {0,1,2}, {3,4,5}, {6,7,8}, {9,10,11}, {12,13,14} };
Array_2[5] [3]={ {15,16,17}, {18,19,20}, {21,22,23}, {24,25,26}, {27,28,29} };
Array[2] [5] [3] = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 ..
.. | 11 | 12 | 14 | 15 | 16 .. | 29;
```

Dieses Schema muss man im Kopf behalten, denn wenn alle 900 LEDs bei 20Hz aktualisiert werden, soll man das Array lieber via Pointerarithmetik traverisieren um Performance zu maximieren. Ansonsten schafft das CPU die Daten nicht schnell genug.

Zum Spieleschreiben bietet sich Objektorientiertes Programmieren hervorragend an. Wird bei Snake ein Pixel von der Schlange gefressen, wird das Pixel Objekt zerstört und dabei z.B. ein Event ausgelöst. Deswegen steht Pixel.h zur Verfügung.

Als Programmieren gibt es also zwei Möglichkeiten das LED Brett zu beschreiben: (1) Durch direktes schreiben des 3D Arrays oder (2) das objektorientiertes Programmieren in dem man Objekt erzeugt und dies mit anderen Objekt interagiert.

5.1.4 Kritischer Bereich und Double Buffering

Ist das 3D Array vollständig beschrieben mit Pixeldaten, dann wird der Vorgang gestartet, endlich ein Bild auf das LED Brett zu bringen. Dafür muss das RMT Peripheral das 3D Array nun ablesen und in elektrischen Spannungen übersetzen. In dieser Zeit darf das 3D Array nicht beschrieben werden. Hat das RMT Peripheral das 3D Array fertig gelesen, wird das Array freigeben und kann wieder beschrieben werden.

Um ein flüssiges Spiel für das menschliche Auge zu gewährleisten, ist ein minimales Frame Rate notwendig. Die Übertragung eines Bits (die Zusammensetzung von T0H+T0L oder T1H+T1L) dauert ungefähr 1,2us. Bei 21600 Bits insgesamt, rechnet sich ein ideales Frame Rate von 33Hz aus. Da das CPU aber verschiedene Programme (Wifi, Webseite, Spiele, usw.) gleichzeitig jongliert, liegt das Frame Rate bei ca. 15 Bilder pro Sekunde. Für ein 1920 x 1080 PC Monitor mit über 2.000.000 Pixeln wären 15 Bilder pro Sekunde sofort bemerkbar. Wenn der Monitor jedoch nur 900 Pixel hat, ist der Vorteil eines flüssigeren Spiels angesichts der hohen Kosten für die CPU-Leistung marginal. Bei 900 Pixeln merkt man keinen Unterschied zwischen einer Bildrate von 30 oder 15.

Das LED Brett setzt Double Buffering ein. So bleibt der kritischer Bereich minimal

klein. Bei einer Bildwiederholungsrate von 15Hz wird alle 70ms ein neues Bild angezeigt. Bevor ein Bild angezeigt wird, muss es beschrieben werden. Es gibt also zwei Phasen zur Erstellung eines Frames: Das Beschreiben des Bildes und Anzeigen des Bildes. Würde beide Phasen über einem Buffer ablaufen, dürfte man nur die Hälfte der Zeit (35ms) das Bild beschreiben, denn in der Zeit wo das Bild angezeigt wird dürfte es nicht angefasst werden. Mit Double Buffering allerdings wird nach jede ca. 35ms das neulich beschriebenes Bild kopiert. Das Anzeigen des Frames läuft dann über diesen zweiten Buffer bzw mit der Kopie, sodass einem das Schreiben eines Bildes praktisch nie gesperrt wird. Diese Strategie klingt toll aber man muss den verfügbaren Systemspeicher in Betracht ziehen. Noch besser wäre dreifaches Buffering, aber das ESP32 hat dafür nicht genug Speicher zur Verfügung.

5.1.5 LED Steuerungsfunktionen und rmtWriteItems (Khan, du kannst diese section nennen, wie du magst) and LED Memory Management

» colorutils.h , dledcontroller.h, dstripdata.h

5.1.6 Flackernde Frames (Lukas)

Während der Nutzung von Arcade LED ist oftmals ein Flackern von mehreren LEDs in verschiedenen Farben auf dem Board für einen Frame sichtbar (siehe Abbildung 7). Dieses Flackern ist erst entstanden, nachdem die LED Board Steuerung mit dem Webserver und Spieldaten gleichzeitig verwendet wurde. Somit ist uns dieses Problem erst sehr spät während der Spieleentwicklung aufgefallen.

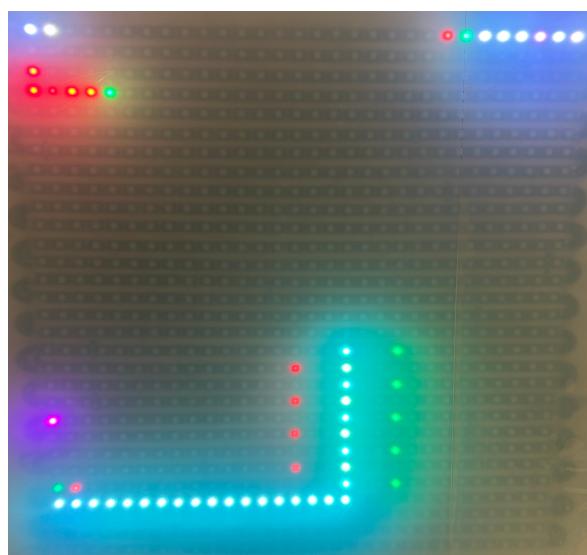


Abbildung 7: Ein von Flackern betroffener Frame des Spiels Snake

Zuerst wurde überprüft, ob der Spieldaten task selbst dem Senden durch z.B. zu frühes Aufrufen von `rmt_write_items()` in die Quere kommt. Allerdings ist dies nicht der Fall, da nach jedem Aufruf der Funktion ein Delay eingebaut ist. Außerdem kann der Funktion eine Flag übergeben werden, durch das erst nach dem vollständigen Senden aus dieser zurückgekehrt wird[1]. In beiden Fällen hat sich nichts am Flackern verändert.

Da der einzige Task, der auf die DLEDControllerklasse zugreift, der Spieletask ist und somit nur dieser allein Daten auf das Board sendet, müssen es die anderen Tasks sein, die dem Senden in die Quere kommen. Durch Testen haben wir herausgefunden, dass das Flackern entsteht, wenn während der Übertragung auf das Board eine Nachricht auf dem Webserver oder Netzwerk eintrifft. Besonders starkes Flackern ist sichtbar, falls eine Website geladen wird.

Wie in folgenden Kapiteln genauer beschrieben wird, wurde der Spieletask auf Kern 1 und alle anderen Tasks, wie der Webserver, auf Kern 0 gemappt. Somit sollten sich die beiden nicht in die Quere kommen. Dies hat allerdings für keine Veränderung am Flackern gesorgt.

Auf dem ESSPRESSIF ESP-IDF Forum gibt es einen Post mit dem genau gleichen Problem. Ein ESP soll eine Reihe an WS2812 LEDs über einen SPIFFSbasierten Webserver ansteuern. Sendet der Webserver eine Datei an einen Client, ist ein Flackern auf den LEDs zu sehen [2]. Die angegebene Lösung für das Problem ist es, das "ESP_INTR_FLAG_IRAM" Flag bei der Installation des RMT Drivers anzugeben [2]. Dieses sorgt dafür, dass die Interrupt Service Routine, die für das Senden des Signals zuständig ist, permanent im Instruction RAM ist [1]. Wäre dies nicht der Fall, müsste möglicherweise die ISR erst aus dem Flash geladen werden [1]. Falls in diesem Moment der Flash allerdings durch das Laden einer Website besetzt ist, könnte dies zu Problemen führen. Für unsere Implementierung hat dies das starke Flackern beim Aufrufen einer Website verringert, aber nicht entfernt.

Mithilfe von Semaphoren wurde sichergestellt, dass das Senden an das Board und Bearbeiten der Requests an den Webserver nicht gleichzeitig stattfinden kann. Dies hat das Flackern vollständig entfernt, aber auch für enorme Ladezeiten auf der Website gesorgt und die Spielsteuerung gestört. Zudem hat es während des Sendens einer Seite zu Standbildern auf dem LED Board gesorgt. Aus diesen Gründen wurde die Semaphoren wieder entfernt.

Trotz Ausprobieren vieler verschiedener Lösungen konnte keine akzeptable für dieses Problem gefunden werden, weswegen immer noch Flackern auf dem LED Board erscheint.

5.2 Backend (Lukas)

5.2.1 Netzwerk

Mithilfe der ESP-IDF API ist es möglich, auf ESP-WROOM Boards das Wi-Fi Modul zu nutzen, um sich mit Netzwerken zu verbinden oder eigene zu erstellen [1]. Für Arcade LED haben wir uns dazu entschieden, ein eigenes Netzwerk auf dem ESP zu verwenden. Somit ist man nicht auf extra Geräte, wie Router, angewiesen, sondern benötigt nur den ESP, das LED Board und eine Stromversorgung.

Die Erstellung des Netzwerks geschieht beim Starten des Systems, bevor die einzelnen Tasks aufgerufen werden. Hier wird auch mithilfe von mehreren API Funktionen das Netzwerk konfiguriert. In unserem Fall wurden bis auf die Kernzuweisung und kleinere Konfigurationen, wie den Namen, die Standardeinstellungen übernommen. Durch die Kernzuweisung auf Kern 0 der zwei Kerne des ESP wurde sichergestellt, dass das Netz-

werk dem Spieldaten, der auf Kern 1 läuft und eventuell ein neues Bild an das Board sendet, nicht in die Quere kommt.

Die Verbindung mit dem Netzwerk ist stabil und selbst mit vier Spielern gleichzeitig schnell genug. Allerdings gab es während der Entwicklung auch Schwierigkeiten. Zu Beginn des Projekts bestanden starke Verbindungsprobleme, die Reichweite und die Stabilität des Signals waren sehr schlecht. Die Ursache hierfür war eine fehlende Antenne auf den ESP-32WROOM-32D Boards [3], die verwendet wurden. Mithilfe einer externen Antenne und später des ESP-32WROOM-32U Boards, welches eine interne Antenne besitzt [3], konnte eine sichere Verbindung hergestellt werden.

Ein weiteres Problem, welches allerdings nicht gelöst werden konnte, ist ein Fehler bei der IP Adressenvergabe. Falls, während ein Gerät mit dem Netzwerk verbunden ist, ein Neustart durchgeführt wird und anschließend nach dem Neustart ein weiteres Gerät sich verbindet, kann es dieselbe Adresse wie das erste Gerät erhalten. Gerät 1 hat allerdings nie den Neustart bemerkt und somit denkt es, die IP Adresse ist noch gültig, kann aber keine Nachrichten mehr empfangen. Da das Problem nur manchmal bei Neustarts auftritt und laut Internetquellen es keine andere Lösung gibt, außer zu warten, bekam dieser Bug eine niedrige Priorität.

5.2.2 Webserver

Um eine einfache Spielsteuerung auf allen Geräten zu ermöglichen, haben wir uns für eine Website entschieden. Diese Website wird auf dem ESP mithilfe der Webserver API der ESP-IDF gehostet. Die API ermöglicht es, selbstgeschriebene Funktionen auf HTTP Requests und URLs zu registrieren [1]. Die Funktionen werden aufgerufen, falls der korrekte HTTP Request mit URL eintrifft.

Wenn man in einem Browser eine URL eingibt, wird von diesem ein HTTP Get Request mit der URL als Inhalt an den Webserver gesendet. Solange dieser Request mit der URL ”/Webapp” beginnt, wird auf unserem Webserver eine Funktion aufgerufen, die aus der URL den Dateipfad ausliest, die korrekte Datei öffnet und diese als Antwort an den Browser zurücksendet.

Die Dateien der Website sind auf dem Flash des ESPs auf einer zweiten Partition mit einem SPIFFS Dateisystem gespeichert. ESP-IDF bietet auch hier APIs an, um solche Dateisysteme zu erstellen und zu verwenden [1]. Dies kommt allerdings mit einigen Limitierungen. Dateinamen sind auf 32 Zeichen beschränkt [1]. Zusätzlich ist dieses Limit schnell erreicht, weil es keine Ordner gibt und diese stattdessen zum Dateinamen hinzugefügt werden [1]. Da ESP32 Code, der nicht explizit im IRAM gespeichert wurde, vor der Ausführung erst vom Flash geladen werden muss, kann es eventuell für Interrupt Service Routinen zu Problemen führen, wenn eine sehr große Datei gesendet wird, in diesem Moment aber die ISR nicht im IRAM ist [1].

Ein weiterer Request Handler reagiert auf die URL ”/Data”. Dieser ist dazu gedacht, Daten über den Zustand der Lobby und der Spieler als Antwort zurückzusenden (siehe Abbildung 8). Somit ist es möglich, auf der Website anzuzeigen, ob bereits ein Spiel läuft oder wie viele Spieler noch benötigt werden, um zu starten.

Jedesmal, wenn der Webserver einen Request erhält, wird eine neue Socket für diesen

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Lobbyzustand (0 - 2)	Spiel ID (0 - 255)	P1 Status (0 - 1)	P2 Status (0 - 1)	P3 Status (0 - 1)	P4 Status (0 - 1)

Abbildung 8: Die Daten, die auf einen HTTP GET Request auf die URL ”/Data” als Antwort zurückgesendet werden.

geöffnet. Allerdings wurden alte Sockets nicht nach Versenden der Datei geschlossen und neu eintreffende Requests ignoriert, da nur eine begrenzte Anzahl an Sockets gleichzeitig geöffnet sein kann. Dies sorgte für sehr lange Ladezeiten, falls mehrere Geräte die Website aufrufen.

Um dies zu umgehen, gibt es eine Konfiguration im Webserver, welche die am wenigsten zuletzt genutzte Socket schließt [1]. Da dies allerdings auch manchmal die Websockets, welche für die Spielsteuerung zuständig sind, zerstörte, konnte es für unser Projekt nicht verwendet werden.

Eigentlich gibt es laut der ESP-IDF Dokumentation eine API Funktion, welche eine Socket nach Vollenden der Aufgabe schließen soll [1]. Da diese allerdings in unserer Version des Frameworks nicht auffindbar war, konnte sie nicht verwendet werden. Stattdessen wird das Error Handling des Webservers ausgenutzt. Falls ein HTTP Request Handler etwas anderes als `ESP_OK` zurückgibt, wird es als Error angesehen und die Socket sofort geschlossen [1]. Dies hat keine erkennbaren Verlangsamungen oder negative Folgen für den Webserver und sorgt dafür, dass selbst mehrere Geräte gleichzeitig die Website laden können.

5.2.3 Spielesteuerung

Auf Arcade LED ist es möglich, mit bis zu vier Spielern gleichzeitig zu spielen. Jeder dieser Spieler ist in der Lage, mit seinem eigenen Gerät Knöpfe zu drücken, welche eine Nachricht an den Webserver senden. Dieser verarbeitet die Nachricht und ändert den Status des korrespondierenden Knopfes des Spielers. Um festzustellen, welcher Spieler gerade einen Knopf gedrückt hat, müssen allerdings die verbundenen Geräte auf die vier Spielerslots gemappt werden.

Die Übertragung der Nachricht wäre z.B. möglich mit HTTP Post Requests. Um sich zu merken, welcher Spieler die Nachricht gesendet hat, könnte eine SpielerID, welche zuvor vergeben wurde und auf dem Gerät des Nutzers gespeichert wird, als Teil der Nachricht mitgesendet werden. Mit dieser Methode könnte es allerdings möglich sein, dass mehrere Geräte dieselbe SpielerID besitzen. Außerdem kann auf dem ESP nicht überprüft werden, welche Spieler noch verbunden sind. Somit können Verbindungsabbrüche nicht registriert werden.

Websockets hingegen ermöglichen es, eine permanente Verbindung zwischen Webserver und Smartphone zu öffnen [1]. Somit können die SpielerIDs auf die Sockets verteilt werden. Die Verbindung der Spieler kann auch leicht über diese überprüft werden, weil im Falle eines Verbindungsabbruchs die Socket geschlossen wird [1].

Da die Websockets für die Spielerverwaltung zuständig sind, muss von der Lobbyerstellung bis zum Spielende alles über diese geschehen. Hierfür gibt es drei Befehle.

Byte 0	Byte 1	Byte 2	Byte 3
Knopfdruck (0)	Knopf ID (0 - 20)	Knopfzustand (0 - 1)	
Lobby erstellen (1)	Spiel ID (0 - 255)	Spieleranzahl (1 - 4)	extra Daten (0 - 255)
Lobby beitreten (2)			

Abbildung 9: Liste der Befehle, die über die WebSocket für die Lobbyerstellung, Lobbybeitreten und Knopfsteuerung gesendet werden können.

Der Lobby erstellen Befehl öffnet eine neue Lobby mit dem ausgewählten Spiel und der Spieleranzahl, falls keine Lobby in diesem Moment existiert. Außerdem wird die WebSocket, über die dieser Befehl erhalten wurde, als Spieler 1 registriert. Anschließend können andere mit dem Lobby beitreten Befehl ihren eigenen Spielerslot erhalten. Sobald die maximale Spieleranzahl erreicht ist, startet das Spiel und mit dem Knopfdruck Befehl kann der Status des ausgewählten Knopfs auf "gedrückt" oder "losgelassen" geändert werden. Somit ist es möglich, auf der Website auch Knöpfe gedrückt zu halten (siehe Abbildung 9).

Wie bereits erwähnt, wird bei Schließung der WebSocket der Spielerslot wieder freigegeben und jemand anderes oder die gleiche Person könnte im Falle eines Verbindungsabbruchs wieder beitreten. Sollten allerdings keine Spieler mehr mit der Lobby verbunden sein, wird diese geschlossen und das Spiel beendet. Anschließend könnte erneut eine neue Lobby geöffnet werden. Somit muss der ESP nie neugestartet werden und es ist möglich, ohne Verzögerung weiterzuspielen.

5.2.4 Spieletask und Taskkommunikation

Da der Webserver und die Spiele parallel ausgeführt werden müssen, existiert eine extra Task für die Spiele auf Kern 1. Solange kein Spiel gestartet ist, zeigt dieser Task den Startbildschirm auf dem LED Board an. Sobald alle Spieler einer Lobby beigetreten sind, wird anhand der SpieleID die Funktion des korrekten Spiels aufgerufen. Wenn das Spiel beendet werden soll, ist es Aufgabe des Spieleanwicklers, alle dynamischen Objekte, die benötigt wurden, zu zerstören und aus der Funktion zurückzukehren.

Die Kommunikation zwischen Spieletask und Webserver erfolgt über ein Objekt der Lobbyklasse, auf das beide Tasks Zugriff haben (siehe Abbildung 10). In dieser Klasse werden alle Daten abgespeichert, wie die Knöpfe der Spieler, welches Spiel gestartet werden soll und der Zustand der Lobby. Da es sich hier um einen kritischen Bereich handelt, muss mit Methoden, die mit Semaphoren abgesichert sind, auf diese Variablen zugegriffen werden.

Um ein neues Spiel für Arcade LED zu entwickeln, muss zunächst eine Funktion mit

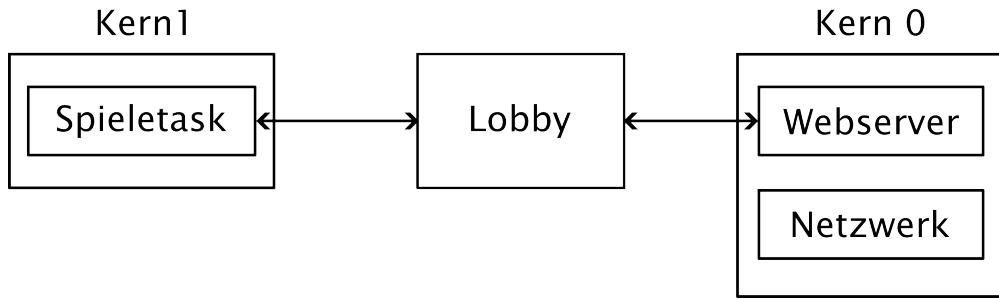


Abbildung 10: Das Diagramm zeigt die Kommunikation über die Lobbyklasse zwischen dem Spiletask und dem Webserver.

Pointern auf die Lobby und die DLEDController Klasse als Parameter erstellt werden. Die DLEDControllerklasse ist nötig, um das LED Board anzusteuern. In der Funktion kann nun das Spiel entwickelt werden. Es muss allerdings sichergestellt werden, dass regelmäßig der Status der Lobby abgefragt wird, da bei dem Zustand "lobby_closed" das Spiel beendet werden muss. Sollte das Spiel normal zu Ende kommen, muss die Verbindung aller Spieler mithilfe einer Methode von Lobby getrennt und auch auf eine Veränderung des Lobbyzustands gewartet werden.

5.3 Frontend (Lisa-Marie)

Besonders wichtig für die Entwicklungsrichtung des Projektes war die Entscheidung über eine grafische Benutzeroberfläche. Diese Entscheidung musste sehr früh im Entwicklungsprozess getroffen werden, da die Konstruktion des Backends sehr stark mit der gewählten Oberfläche gekoppelt ist. Die Kommunikation über eine geeignete Schnittstelle muss gut durchdacht sein und die Ansteuerung des LED-Boards ist auf die Grafische Benutzeroberfläche angewiesen, welche die Verbindung zu dem Nutzer darstellt.

5.3.1 Vorüberlegungen

Das Projekt Arcade LED ist die Neuauflage eines früheren Projektes und soll die Schwachstellen der Vorgängergruppe verbessern. Einer der wichtigsten Aspekte des Projektes war die Verbesserung der Kommunikation zwischen den Nutzern und dem LED-Board. Unsere Vorgänger nutzen eine Android App für die Spieldatenwahl und Steuerung. Leider funktionierte die Steuerung sehr holprig und das Erscheinungsbild/Design der Applikation war sehr kontraintuitiv für die Spieler.

So stellt sich die Frage nach der geeigneten Oberfläche, welche sowohl effektiv in der Kommunikation mit dem Microkontroller als auch einfach in der Benutzung für Spieler ist. Die möglichen Optionen waren auch in unseren Überlegungen entweder eine App oder eine Webseite.

Nachteile der App:

- Plattformabhängigkeit: eine App müsste für die verschiedenen Betriebssysteme separat entwickelt werden und hätte somit einen deutlich höheren Entwicklungsaufwand und potenzielle Kosten. Sich nur auf ein Betriebssystem zu einigen widerspricht unseren Anforderungen für das Projekt.
- Installation und Aktualisierung: Diese Apps müssten in den App Stores der jeweiligen Betriebssysteme bereitgestellt und für die Installation verfügbar sein. Bei Anpassungen der Applikation müssten stetige Updates über den App Store ermöglicht werden. Zudem müsste das Endgerät über ausreichend Speicherplatz verfügen.
- Kommunikation: Die Schnittstelle der Kommunikation zwischen einer App und einem Microkontroller ist mit unserem ESP32 aufwendiger. Hinzu kommen mögliche Kompatibilitätsprobleme.

Vorzüge der Webseite:

- Plattformunabhängigkeit: Eine Webseite kann auf verschiedenen Betriebssystemen und Geräten genutzt werden, da sie über den Webbrower zugänglich ist. Dadurch entfällt die Notwendigkeit, separate Versionen für verschiedene Plattformen zu entwickeln.
- Wartung und Updates: Änderungen an einer Webseite können zentral vorgenommen werden, da sie auf einem Server gehostet wird. Dadurch entfällt die Notwendigkeit, Updates für jede einzelne Installation zu veröffentlichen.
- Kommunikation: Der Austausch von Daten mit einem ESP32 kann mithilfe einer guten WebSocket-Implementierung stattfinden. Für nähere Definition siehe Abschnitt ??.

Die Wahl zur grafischen Benutzeroberfläche fiel hiermit auf die Webseite, da vor allem in den Anforderungen für unser Projekt keine Festlegung auf ein einziges Betriebssystem gewünscht ist. Die Webseite muss nun in der Lage sein auf den gängigen Wegen via Http erreichbar zu sein. Natürlich muss zusätzlich der grundlegende Zweck zur Steuerung und Spielauswahl erfüllt werden. Die Webseite soll Eingaben von dem jeweiligen Endgerät an den ESP32 senden, der diese sodann an das LED-Board weiterreicht.

5.3.2 Struktur und Aufbau

Die Konstruktion einer Webseite ist für mich der erste Berührungsypunkt mit Frontend-Entwicklung und somit war ein wenig Vorlaufzeit für die Recherche und Informationsbeschaffung nötig. So fiel meine Wahl schließlich auf eine *Multi-Page-Applikation* (MPA). Bei der MPA handelt es sich um die „klassische“ Variante einer Website, wobei jede einzelne Seite (im folgenden „Page“ genannt) ihren eigenen URL besitzt. Dieser kann kopiert und in das Browsersuchfeld eingefügt werden. Die Navigation erfolgt mit Referenzierungen der nächsten Pages und den Vor-/Zurückpfeilbuttons.

Zwar ist die MPA durch häufiges Laden und Rendern der neuen Pages teilweise langsamer als der Konkurrent die „Single-Page-Applikation“, aber das kann für unseren Use Case vernachlässigt werden. Der Overhead durch das Laden der Pages ist für die Spielerfahrung irrelevant, da im Steuerungsbereich der Spiele keine Pages geladen werden müssen.

Zusätzlich ist die MPA insofern von Vorteil, da für Mehrspieler Spiele eine Art Wartebereich, „Lobby“, benötigt wird. Diese Lobby kann von einer Page aus gesteuert werden ohne die Navigation zu stören. Im Umkehrschluss kann die Erstellung der Lobby „abgeschottet“ werden und keine weiteren Störfaktoren zugelassen werden (genaueres in späterem Kapitel).

Die Webseite ist klassisch mit den Sprachen HTML, CSS und Javascript geschrieben ohne besondere Frontend Programm-Bibliotheken. Das finale Format besteht aus vier Seiten, drei CSS-„Stylesheets“ und einer Javascript-Datei für die Kommunikation mit dem Backend. Die Webseite ist strikt auf eine Nutzung mit Displays, welche eine Beührungssensor besitzen, ausgelegt wie sie für Tablets oder Smartphones üblich sind.

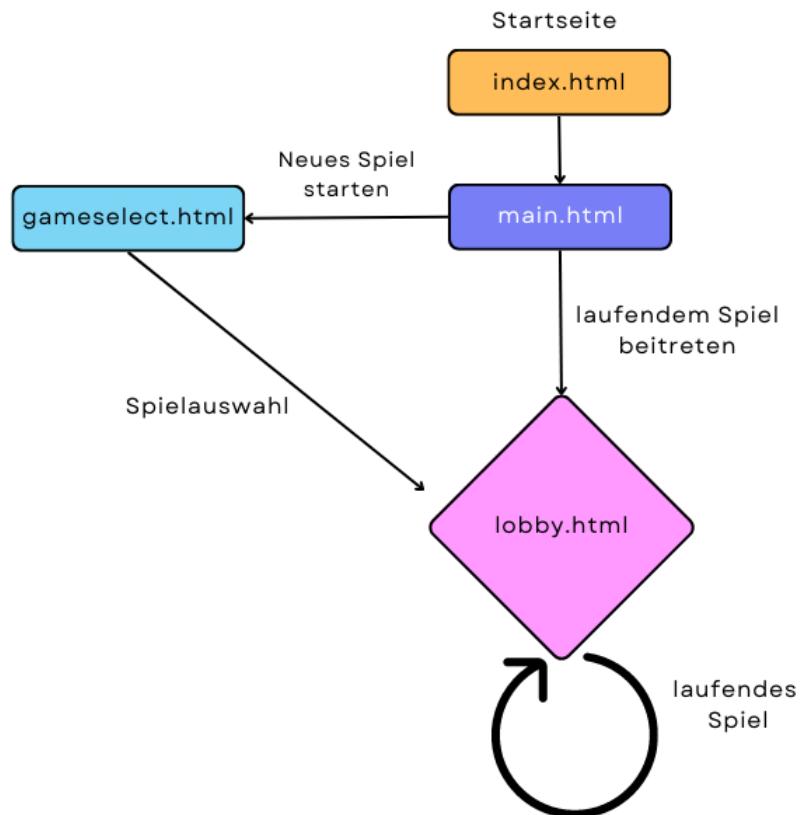


Abbildung 11: Flussdiagramm zur Navigation durch die Webseite

Wie das Diagramm in Abbildung 11 darstellt, referenzieren sich die Webseitenpages gegenseitig um einen simplen und benutzerfreundlichen Umgang zu garantieren. Die Navigation sollte einfach sein und ein intuitives Umfeld erzeugen. Die Größe der Webseite mit allen Dateien umfasst grob 30kB und ist ebenfalls speicherfreundlich für den begrenzten Speicher des ESP32. Die Pages befinden sich in den Spiffs-Partitionen des ESP.

Die Webseite befindet sich unter „data/Webapp“ und die einzelnen URL's bauen sich wie folgt zusammen „Netzwerkmaske/Webapp/Page“.

5.3.3 Seitenspezifikationen

Dieser Abschnitt beschreibt die einzelnen Pages und ihre Besonderheiten. Alle Seiten sind UTF-8 zeichenkodiert und entsprechen den restlichen Standards heutiger HTML Dokumente.

1. Index.html:

Diese Page ist der Ausgangspunkt der Anwendung. Für die leichtere Bedienung ist ein QR-Code mit dem URL der Startseite verfügbar. So müssen die Nutzer nicht den gesamten URL in den Browser eingeben. Die Seite hat keine weitere Spezifikationen und leitet den Nutzer bei einem Klick auf den Hauptbutton zur nächsten Page weiter. Diese Seite soll den Charakter des LED-Boards und der Spiele einfangen. Das Design des Buttons soll an eine bunte Spielhalle erinnern und bei Aktivierung erscheint ein Leuchtkranz. (genaueres in der Beschreibung für CSS-Dateien unter Kapitel 5.3.4 „Design und Kommunikation“)



Abbildung 12: Ansicht der Startseite „index.html“ der Webanwendung

2. Main.html:

Diese Page nimmt eine Schlüsselrolle ein. Sie erlaubt die erste Wahl zwischen einem neuen Spiel und dem Betreten („join“) einer bereits erstellten Lobby. Die Wahl des neuen Spiels leitet den Nutzer weiter zur Spieldatenwahl. Da die Spieldatenwahl bereits wichtige Zwischendaten in den lokalen Speicher der Webanwendung legt, ist es nicht möglich auf die Seite der Spieldatenwahl zu gelangen, wenn bereits ein Spiel im Gange ist oder eine Lobby existiert.

Ist dies der Fall, so wird ein Informationsfeld eingeblendet, welches dies mitteilt, zu sehen in Abbildung 13. Das Betätigen des Buttons ist erst dann wieder möglich, wenn das aktuell laufende Spiel oder die gestartete Lobby beendet wurde. Der Status kann durch Aktualisieren der Seite abgefragt werden.

Wird die Option „Einem Spiel beitreten“ gewählt, so wird der Nutzer auf die Page „lobby.html“ weitergeleitet. Nutzer werden in die Rolle Host und Mitspieler aufgeteilt. Je nach Einordnung bekommen sie verschiedene Versionen der Lobby zu sehen. (genaueres unter Lobby.html)

3. Gameselect.html:

Diese Page ist für das Setzen der GameID, Spieleranzahl und Zusatzdaten für das Spiel „Othello“ verantwortlich. Um Nebenerscheinungen zu vermeiden ist die Spieldatenwahl ab dem Zeitpunkt der Erstellung einer Lobby nicht möglich. Alle



Abbildung 13: Ansicht der Page „main.html“ der Webanwendung

Buttons führen bei Klick das Setzen der notwendigen Daten in den lokalen Speicher der Webanwendung aus. Hierbei kann einem Key ein String-Wert zugewiesen werden, worauf von anderen Pages heraus zugegriffen werden kann. Bei Abgreifen der Werte muss mit der Funktion „parseInt()“ der Wert der Integer unter dem Reiter Value konvertiert werden, wie in folgender Grafik zu sehen ist.

Einträge durchsuchen	
Key	Value
GameID	3
Host	true
Lobbybool	false
MaxPlayer	2
Player	false

Abbildung 14: Inhalt des lokalen Speichers der Webanwendung

Die Arbeit mit dem lokalen Speicher war notwendig, da die Erstellung der Lobby mit einer einzigen Funktion ablaufen soll. Diese Funktion muss alle notwendigen Daten, wie GameID und Spieleranzahl an das Backend via Websockets übertragen um ein korrektes Starten der Spiele zu gewährleisten. Eine Eigenheit der Websockets ist jedoch, dass sie nur für die aktuelle Page eine Verbindung offen halten. Sobald also eine Seite durch Betätigen eines Buttons („href“ genannt) weitergeleitet wird, schließt sich auch der WebSocket.

Zur anfänglichen Entwicklung wurde die Lobby direkt bei Klick auf den Spielbutton gestartet. Diese hat dann durch die href-Weiterleitung auf die Controller-Page geführt. Jedoch schloss sich durch den href die Socket-Verbindung und die Lobby wurde direkt wieder beendet. Die Umgehung dieses Problems war dann der lokale Speicher, hier werden GameID und MaxPlayer gesetzt, welche dann in der

Lobby.html abgerufen und mit dem WebSocket an das Backend geschickt werden kann ohne das Beenden der Lobby zu erzeugen.

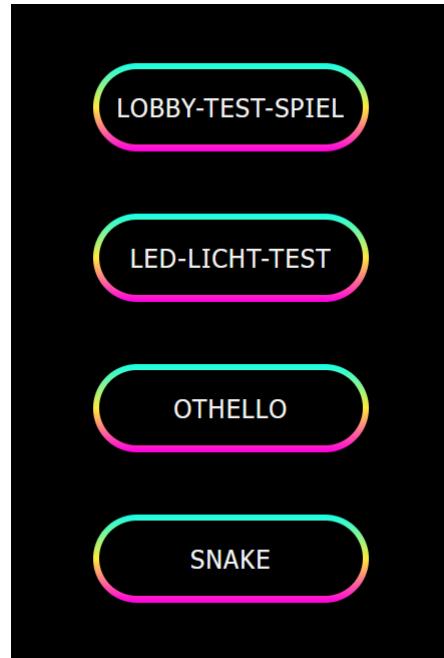


Abbildung 15: Ansicht der Page „gameselect.html“ der Webanwendung

Zudem leiten die Buttons auf die Page „lobby.html“ wo die Lobby gestartet wird und die Spielecontroller zu finden sind. Eine kleine Abweichung gibt es bei der Wahl für Othello, wobei durch Klicken des Buttons eine Auswahl an Karten auf dem Bildschirm erscheint. Erst nach Wahl der Spielkarte werden GameID und Maxplayer gesichert. Zusätzlich wird hier noch der Wert „Map“ gesichert, welcher ansonsten leer bleibt. Dieser wird bei der Erstellung der Lobby berücksichtigt.

4. Lobby.html:

Auf dieser Page wird das gewählte Spiel gestartet und die Ansicht auf die Spielsteuerung umgeschalten. Um sicherzustellen, dass die WebSocketverbindung während des Aufenthalts des Users auf dieser Seite stabil bleibt, kann (aus oben genanntem Grund der WebSocket-Eigenheit) keine separate Seite für die Spielbedienung benutzt werden, da sonst die Lobby direkt wieder geschlossen werden würde.

Die Lobby.html ist insofern besonders, da sie drei verschiedene Ansichten besitzt. Die Host-Ansicht, welche sich den Nutzern zeigt die über die Gameselect.html ein Spiel ausgewählt haben (siehe Abbildung 15). Die Mitspieler-Ansicht, welche den Nutzer gezeigt wird die von der Main.html einem bereits gestartetem Spiel beitreten wollen.

Die Sicherung dieser zwei Ansichten erfolgt über einen sogenannten „Document Referrer“, welcher sich den URL der vorher besuchten Page sichert. Es werden zwei Werte (Host, Player) in den lokalen Speicher geladen und jeweils wahr oder falsch gesetzt. Ausgehend von diesen Werten werden die Ansichten für den Host aktiviert und die der Spieler deaktiviert, vice versa.

Zuletzt gibt es noch die Spielcontroller-Ansicht welche die Buttons für die Spiel eingaben, wie Pfeiltasten und A/B-Knöpfen, bereitstellt. Für die Betätigung der Knöpfe wurde ein „Eventlistener“ für Berührungsereignisse, wie Fingerdruck, erstellt. Durch diese Eventlistener kann die Berührung und das folgende Loslassen genau detektiert werden. Somit müssen die Knöpfe nicht entprellt werden. Diese Listener beinhalten die Funktionen „ontouchstart/end“, welche durch Betätigung der Knöpfe aktiviert werden. Bei jedem dieser Events wird eine Funktion zur Übermittlung der Information, welche der Knöpfe gedrückt(1) oder losgelassen(0) wurde, über die bestehende WebSocketverbindung an den Microkontroller aufgerufen.

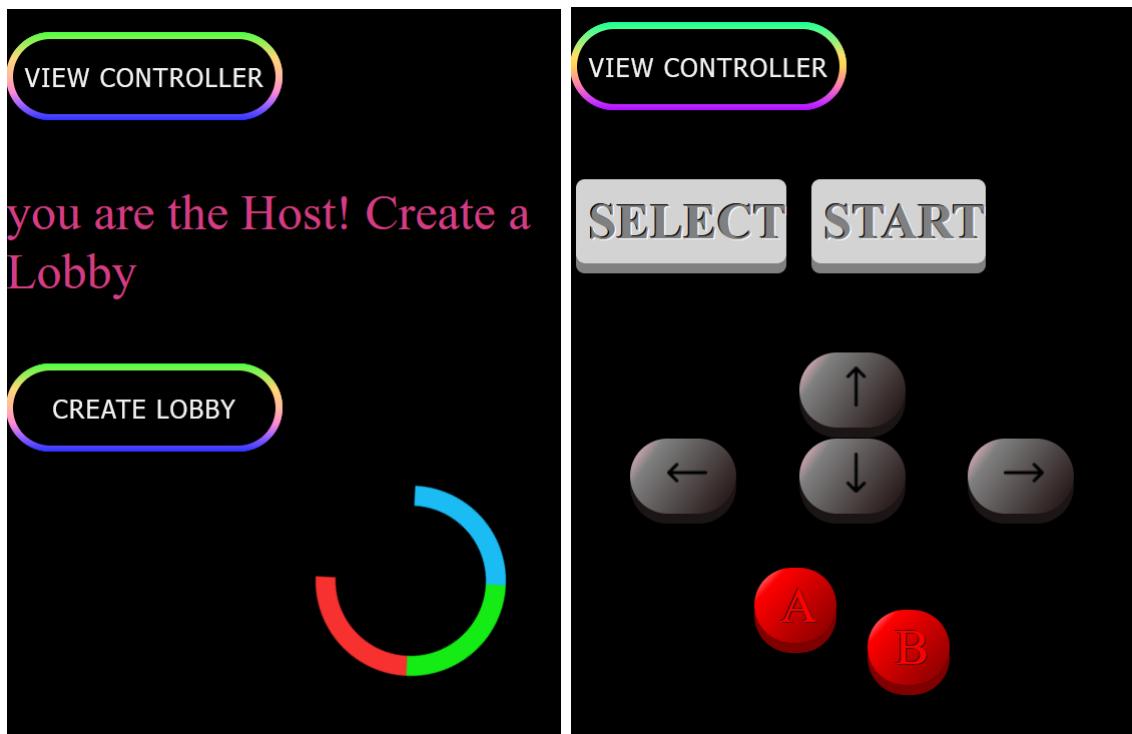


Abbildung 16: Host-Ansicht(links) und die Spielsteuerung(rechts) eingebettet in die „lobby.html“

Der Ring in der rechten unteren Ecke stellt einen Lade-Ring dar und sobald der Button „Create Lobby“ benutzt wurde, wird der Wert Lobbybool im lokalen Speicher auf „true“ gesetzt. Der Ring wird zu einem grünen Haken und zeigt somit grafisch, dass die Lobby erstellt wurde und dient der leichteren Bedienbarkeit für Nutzer. Zeitgleich wird die Funktion Create Lobby aufgerufen und eine Verbindung zum Backend hergestellt. Ab diesem Zeitpunkt kann dem Spiel beigetreten werden.

Die Ansicht für Mitspieler gleicht dem des Hosts, bis auf den Button „Create Lobby“. Hier wird stattdessen ein „Join Game“ Button zur Verfügung gestellt. Wird der Button betätigt obwohl keine Lobby geöffnet ist, so passiert nichts. Dieser Fall wird in der Auslese-Funktion der Websockets abgefangen.

Sind alle notwendigen Spieler beigetreten, also die Anzahl an verbundenen Spielern der Anzahl an MaxPlayer des gewählten Spiels entspricht, wird der Startbild-

schirm des jeweiligen Spiels auf dem LED-Board angezeigt. Alle Spieler können nun durch Betätigen des „View Controller“ Buttons auf die Spielsteuerung zugreifen. Ein wiederholtes Betätigen führt zurück zur Host-/Spieleransicht. Das Erscheinenlassen der einzelnen Ansichten erfolgt durch das geschickte hinzufügen der CSS-Klasse „hidden“ wodurch Page-Elemente alle gleichzeitig existieren können jedoch nicht angezeigt werden. Nicht-anzeigte Elemente können nicht betätigt werden.

5.3.4 Design und Kommunikation

Das Erscheinungsbild der Website hat ebenfalls einen hohen Stellenwert eingenommen. Das Design der Buttons in dem Arcade Licht Stil soll bereits zum Betreten der Startseite Lust auf das Spiel bereiten. Im Folgenden werden die Stylesheets, welche das Aussehen und die Designelemente der einzelnen Pages erzeugen, beschrieben. Zudem wird die Datei „Lobby.js“ näher erklärt, da diese die Funktionen für die Kommunikation mit dem Backend koordiniert.

- CSS-Stylesheets:
 1. Arcade.css: Dieses Stylesheet ist die Grundlage mit dem Design für die großen bunt rotierenden Buttons, welche für die Webseite verwendet werden. Grundsätzlich ist das Design der Webseite ein schwarzer Hintergrund, so dass die leuchtenden Farben der Buttons und Knöpfe zur Geltung kommen. Die CSS-Datei wird von jeder Page eingebunden und zur Zeit des Http-Requests vom ESP versendet.

Die Buttons werden durch mehrere Überlagerungen von Stilelementen erzeugt. Dabei werden zwei Klassen verwendet „inner/outer“ welche erstens die ovale Form und zweitens die Farben beinhalten. Das innere Objekt ist farbbestimmend und das äußere rotierend. Mithilfe von sogenannten „Span“-Elementen wird ein Schimmer um den Button erzeugt, welcher bei Betätigen der Buttons aufleuchtet und ihn umgibt. Dabei müssen erneut zwei dieser Span's verwendet werden, da der Button ebenfalls aus zwei übereinander gelagerten Objekten besteht.

Zusätzlich wird auch noch das Popup, welches in der Main.html die Information über bereits laufende Spiele enthält, verwaltet. Hierbei wird ein einfacher grauer Hintergrund mit weißer Schrift verwendet. Das Objekt ist „hidden“ und wird erst nach Abfrage auf einen Wert angezeigt.

2. Controller.css: Diese Design-Komponente wird lediglich von der Lobby eingebunden. Sie ist für die Darstellung der Knöpfe, welche zur Spielsteuerung notwendig sind, zuständig. Insgesamt existieren acht Knöpfe: vier Richtungspfeile, zwei Knöpfe A und B sowie die Buttons Start und Select. Das Design der Buttons ist an das klassische Layout der Nintendo-Konsolen angelehnt. Die Design-Idee stammt von einem Hobby-Entwickler [4] und wurde für das Projekt modifiziert.

Heutige Mobilebrowser, wie beispielsweise Apple Safari, besitzen die Funk-

tion, sämtliche Objekte in einer Webumgebung kopieren und deren Text extrahieren zu können. Sobald ein längeres Halten des Fingers auf dem Touchscreen erkannt wird, beginnt das Interface mit dem Einhüllen des Objektes, anstatt es zu Betätigen. In dem Fall der Spielsteuerung wurde statt der Betätigung der Knopf selbst kopiert, welches den Spielfluss, die Bedienung und das Spielerlebnis beeinträchtigt hat. Daher musste für die einzelnen Knopf-Klassen die Kopie untersagt werden mit beispielsweise „-webkit-user-select: none“.

3. **Lobby.css:** Das dritte Stylesheet bestimmt das Design der Lobby-Komponenten: der Ladering, der Bestätigungshaken und ein grüner Pfeil.
- **Lobby.js:**
 Die Javascript-Datei ist für die Kommunikation mit dem Backend zuständig. Die enthaltenen Funktionen setzen elementare Aktionen, wie das Erstellen „createLobby()“ und das Beitreten „joinLobby()“ einer Lobby sowie die Buttonbetätigung „Button()“ um. Zusätzlich enthält die Datei einen „Http-Request-Listener“ welcher für das Auslesen von Daten aus dem Backend benötigt wird. Hierzu ist auch die Requestfunktion „getData()“ notwendig.
 - **createLobby():** Diese Funktion gibt den Befehl an das Backend eine Lobby zu erstellen, falls noch keine geöffnet wurde. Die Funktion öffnet einen Websocket und erstellt ein Botenarray der Länge vier mit den notwendigen Daten. Hierbei sind der Kennzeichner für die Aktion „Erstellen“ einer Lobby (1), die GameID, die maximale Spieleranzahl und eine Map für das Spiel Othello enthalten. Die GameID und Spieleranzahl werden aus dem lokalen Speicher der Webanwendung geladen und zu Integerwerten umgewandelt. Die Map wird nur im Fall des Spiels Othello aus dem lokalen Speicher geladen und ist ansonsten Null.
 - **joinLobby():** Das Beitreten wird sehr simpel umgesetzt. Hier wird lediglich ein Websocket geöffnet und der Kennzeichner (2) verwendet. Das Datenarray ist eлементig, da keine weiteren Daten nötig sind. Wird die Funktion aufgerufen, obwohl keine Lobby geöffnet ist, passiert nichts.
 - **Button(a):** Diese Funktion nutzt ein Botenarray der Länge drei. Zuerst der Kennzeichner für Knopfdruck (0), gefolgt von der KnopfID und der Information, ob jeweils betätigt oder losgelassen wurde.
 - **getData() und reqListener():** Die Funktion getData() wird benötigt um auf Änderungen und gesetzte Werte des Backends zu reagieren, beispielsweise ob eine Lobby bereits aktiv ist. Es wird ein „XMLHttpRequest“-Objekt erstellt, welches das Senden aktueller Lobbydaten anfordert.

Die zurückgesendeten Daten werden von dem reqListener() abgefangen und in eine Variable gespeichert. Die Antwort beinhaltet Informationen über den Status der Lobby, der GameID und den verbundenen Spielern. Bisher wird in der Funktion lediglich der Lobbystatus abgefragt. Die Funktion getData() wird beim Betreten der Page Main.html aufgerufen und basierend auf dem Status wird der Button inaktiv und das Informationsfeld über eine laufende

Lobby wird eingeblendet.

5.3.5 Einbindung neuer Spiele

Dieses Kapitel enthält Informationen über die notwendigen Schritte um ein neues Spiel von Sicht der Webseite einzubinden. Die Erweiterung des Spielangebots sollte so simple und einfach wie möglich umgesetzt werden, weshalb nur sehr wenige Schritte notwendig sind.

1. Spielauswahl:

Um ein neues Spiel für die Nutzer sichtbar zu machen, muss in der Gameselect.html ein neuer Button hinzugefügt werden. Für das Spiel muss eine „setState()“ Funktion und die „ChangeRef()“ aufgerufen werden. Die Erste, um spielspezifische Daten in den lokalen Speicher zu sichern und die Zweite um die href auf die Lobby auszuführen.

2. Falls weitere Daten für ein Spiel notwendig sind, muss die Funktion zur Erstellung der Lobby an die Anzahl an Informationen angepasst werden.
3. Die Spielsteuerung muss nicht angepasst werden, da diese auf die Lobby abgestimmt sind. Wenn andere Buttons benötigt werden als bisher vorhanden, müssen diese verständlicher Weise hinzugefügt werden.

5.3.6 Idle-LED-Anzeige

Die Anzeige auf dem LED-Board beginnt erst zu leuchten, wenn der Idle-Status des Microkontrollers durch das Starten eines Spiels verlassen wird. Wenn also die Lobby vollständig und alle Spieler beigetreten sind, wird der Startbildschirm des jeweiligen Spiels angezeigt. Aus Gründen der Optik und auch für die bessere grafische Darstellung des Spielstatus wurde eine Idle-Anzeige hinzugefügt.

Diese befindet sich in der Main-Funktion des Quellcodes unter „IdleScreen(DLEDController * LED)“. Der GameTask der Main durchläuft eine Endlosschleife und solange keine Lobby, beziehungsweise Spiel, gestartet wurde, wird die Funktion aufgerufen. Sie besteht aus zwei großen For-Schleifen, welche den gesamten Spielfeldrand ablaufen. Die erste Schleife setzt die Rot-Blau-Grün-Werte der einzelnen LED's und die Zweite entfernt bestimmte Farbwerte wieder. Durch das Hinzufügen und Entfernen bestimmter Farbanteile entsteht ein optisch schöner Farbverlauf.

Durch den Farbverlauf ist zu erkennen, dass sich der Microkontroller noch immer im Idle-Status des Haupttask befindet und bisher kein Spiel gestartet wurde.

5.4 Spiele

Dieses Kapitel beschreibt die beiden Spiele, ReversiXT und Snake, welche für das Board verfügbar sind.

5.4.1 Othello/ReversiXT (Lukas)

Othello ist Brettspiel, bei dem zwei Spieler abwechselnd Steine setzen und Steine des Gegners einnehmen. ReversiXT ist eine von Prof. Dr. Carsten Kern entwickelte Erweiterung. Im Folgenden wird auf die genauen Regeln des Spiels und die Implementierung eingegangen.

5.4.1.1 Regeln

Othello wird auf einem 8x8-Brett mit zwei Spielern gespielt [5]. Zu Beginn des Spiels besitzen beide zwei Steine in der Mitte des Spielfeldes [5]. Die Spieler setzen abwechselnd einen Stein auf ein freies Feld, welches horizontal, diagonal oder vertikal an ein bereits besetztes Spielfeld angrenzt [5]. Alle gegnerischen Steine, die zwischen einem eigenen und dem gelegten Stein liegen, werden daraufhin in die eigene Farbe umgedreht [5] (siehe Abbildung 18). Ein Zug ist nur gültig, wenn dabei gegnerische Steine eingenommen werden [5]. Falls ein Spieler nicht ziehen kann, darf der andere erneut setzen [5]. Sollten beide keine Steine einnehmen können, wird das Spiel beendet und der Spieler mit den meisten Steinen gewinnt [5].

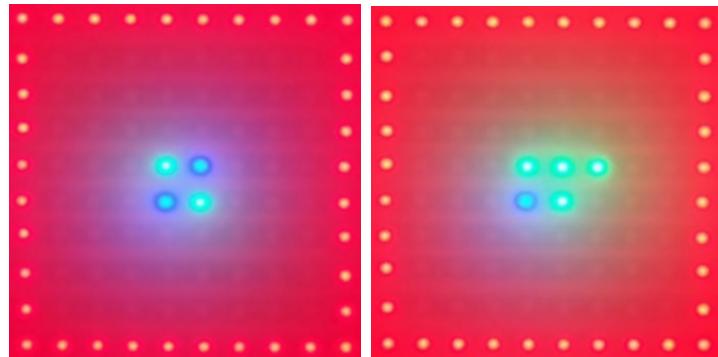


Abbildung 17: Das linke Bild zeigt ein Othellospiel in der Startaufstellung. Im rechten Bild sieht man dasselbe Spiel, nachdem der grüne Spieler einen Stein rechts neben den oberen rechten Stein des blauen Spielers gesetzt hat.

ReversiXT ermöglicht es, mit mehr als zwei Spielern zu spielen [6], in dieser Implementierung bis zu vier. Da die Spieleranzahl variieren kann, gibt es auch unterschiedlich große Spielfelder. Diese können sich auch in ihrer Form zu dem Standarddreieck in Othello unterscheiden [6].

Zusätzlich gibt es vier verschiedene Arten von Sonderfeldern [6]. Falls ein Spieler einen validen Zug auf diese ausführt, werden die Effekte dieser einmalig aktiviert [6]. Das Wahlfeld ermöglicht es, mit einem Spieler seiner Wahl die eingenommenen Steine zu tauschen [6]. Das Inversionsfeld tauscht die eingenommenen Steine aller Spieler miteinander [6]. Das Bonusfeld gibt dem Spieler die Wahl zwischen einem Überschreibstein

oder einer Bombe [6]. Expansionsfelder verhalten sich wie gegnerische Felder und können somit eingenommen werden [6].

Überschreibsteine sind eine begrenzte Ressource, die es einem erlauben, bereits eingenommene Spielfelder erneut zu besetzen, solange es sich trotzdem um einen validen Zug handelt [6]. Bomben können in der 2. Phase des Spiels verwendet werden [6]. Sie beginnt, nachdem kein Spieler mehr einen validen Zug durchführen kann [6]. In dieser Phase werden anstatt Steine Bomben gesetzt, welche alle Felder in deren Umkreis zu nicht besetzbaren Wänden ändern [6]. Es wird solange weiter gezogen, bis alle Bomben aufgebraucht sind oder das Spielfeld komplett zerstört ist [6]. Auch hier ist der Gewinner der, der am Ende die meisten Steine besitzt.

Eine weitere Regel von ReversiXT sind Transitionen. Diese ermöglichen es, Felder, die an Wände angrenzen, miteinander zu verbinden und somit Züge durch Wände hindurchzuführen [6]. Da dies allerdings nicht sinnvoll auf dem LED Board dargestellt werden kann, wurde diese Regel entfernt.

5.4.1.2 Implementierung

Wie wird solch ein Spiel auf einem 30 * 30 LED Board angezeigt und mit einfachen Knöpfen kontrolliert? Die naheliegendste Lösung ist ein Cursor, welcher sich mithilfe der Pfeiltasten bewegen lässt. Mit der A Taste kann ein Stein auf die Koordinaten des Cursors gelegt werden. Somit ist es möglich, Züge auszuführen. Da es sich aber um ReversiXT handelt, können Karten deutlich größer als das LED Board sein. Um dies zu umgehen, wird nicht der Cursor, sondern das ganze Bild bewegt (siehe Abbildung 18).

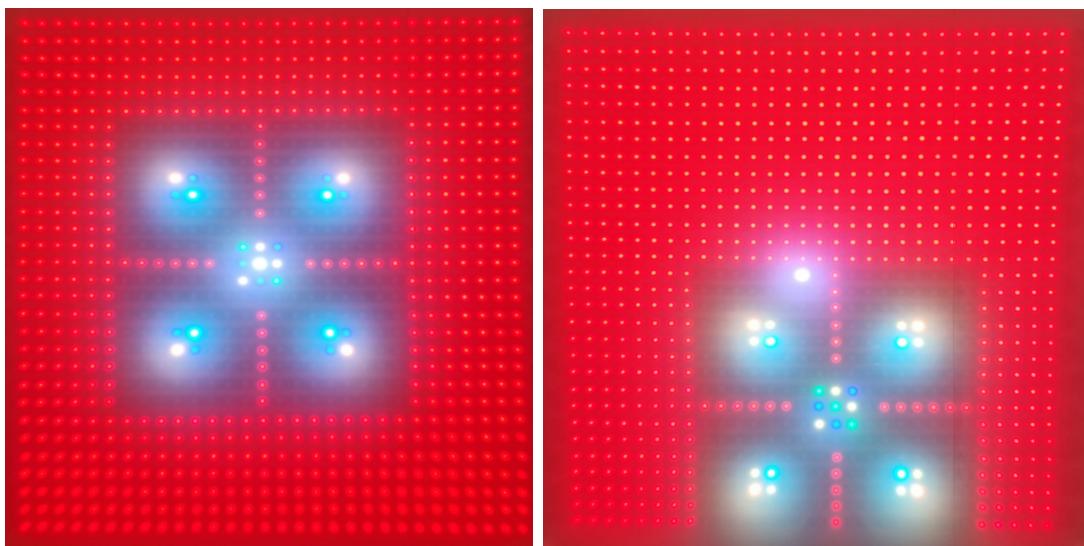


Abbildung 18: Auf dem linken Bild befindet sich das Spielfeld in der Mitte des LED Boards. Der Cursor wurde auf dem rechten Bild nach oben bewegt, aber bleibt konstant in der Mitte des Boards stehen. Stattdessen wurde die Karte nach unten verschoben.

Für Wahl- und Bonusfelder wird ein extra Bild angezeigt, auf dem man auch mit den Pfeiltasten und dem A Knopf seine Entscheidung treffen kann. Die Farbe der Rahmen zeigt an, welcher Spieler gerade am Zug ist (siehe Abbildung 19).

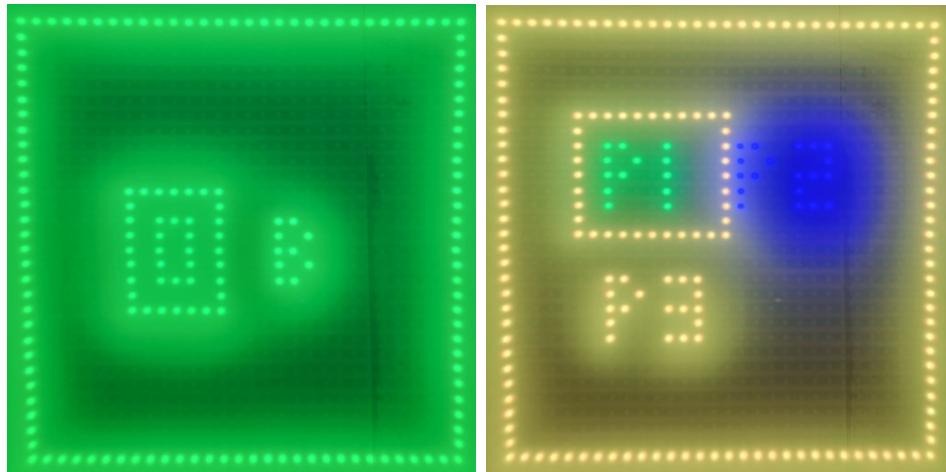


Abbildung 19: Das linke Bild zeigt die Auswahl zwischen einem Überschreibstein (O) und einer Bombe (B), das rechte die eines Wählfeldes. Bei beiden leuchtet der große und kleine Rahmen in der Farbe des Spielers, der die Entscheidung trifft. Der kleine Rahmen markiert, was gerade ausgewählt ist.

Um zu erkennen, welcher Spieler gerade dran ist, wird vor jedem Spielzug der Spieler in seiner Farbe mit der Anzahl der Überschreibsteine und Bomben angezeigt (siehe Abbildung 20). Dieses Bild bleibt solange auf dem Board, bis irgendeine Taste des Spielers gedrückt wird.

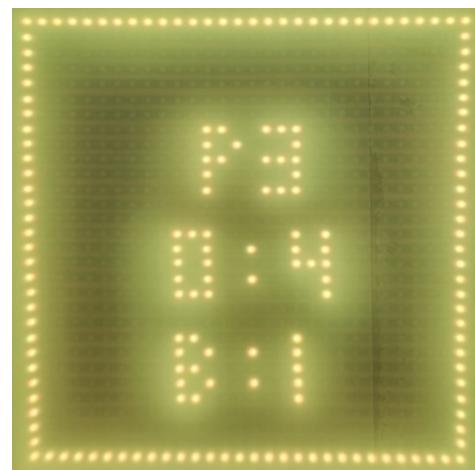


Abbildung 20: Dieses Bild wird vor jedem Spielzug in der Farbe und mit den Daten des Spielers, der nun am Zug ist, angezeigt. In diesem Fall handelt es sich um Spieler 3 mit vier Überschreibsteinen (O) und einer Bombe (B).

Da das LED Board, wegen Hardware Problemen oder weil die Schnittstelle noch in Entwicklung war, nicht immer verwendet werden konnte und zudem das Entfernen des ESP vom LED Board, das Flashen und anschließende Wiederanstecken beim Debuggen sehr viel Zeit in Anspruch nahm, wurde häufig die serielle Schnittstelle verwendet. Mithilfe dieser konnte das gesamte Spielfeld auf der Kommandozeile mit ASCII Zeichen angezeigt werden und somit bis auf die richtige Bildausgabe alles getestet werden (siehe Abbildung 21). Allerdings sollten währenddessen die Debug Nachrichten in anderen Tasks wie dem Webserver ausgeschaltet werden. Diese könnten sonst einige Frames

zerstören, was bei einem Spiel wie Othello keine großen Probleme bereitet, aber bei einem schnelleren Spiel wie Snake schon.

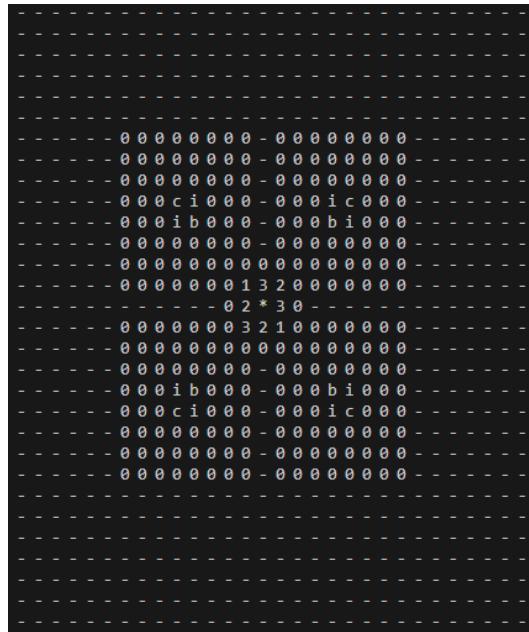


Abbildung 21: Kommandozeilenausgabe der 3 Spieler Karte aus Abbildung 18 für Debugging.

ReversiXT ist beliebig mit neuen Karten erweiterbar. Eine neue Karte kann mithilfe von einfachen ASCII Zeichen in einer txt Datei erstellt werden. In der ersten Zeile kommt die Anzahl der Überschreibsteine. Die zweite Zeile enthält die Anzahl der Bomben und deren Explosionsradius. Danach folgen die Höhe und Breite der Karte und zuletzt kommt die Karte Zeile für Zeile. Diese Datei muss auf das SPIFFS Dateisystem in den Othelloordner geflasht und in der loadmap Funktion unter eine neue Map ID in das switch eingefügt werden.

Die Farben, die für das LED Board verwendet werden, und die ASCII Zeichen, die für die Kartenerstellung und serielle Schnittstellenausgabe genutzt werden, sind aufgelistet in folgender Abbildung (22).

Farbe	ASCII	Feldtyp
keine Farbe	0	leeres Feld
	1	Spieler 1
	2	Spieler 2
	3	Spieler 3
	4	Spieler 4
	-	Wand / außerhalb der Karte
	i	Inversionsfeld
	c	Wahlfeld
	b	Bonusfeld
	x	Expansionsfeld (auf Karten nicht verwendet)
	*	Cursor

Abbildung 22: Die Farben, die für das LED Board verwendet, und die ASCII Zeichen, die für die Kartenerstellung und serielle Schnittstellenausgabe genutzt werden.

5.4.2 Snake (Lisa-Marie)

Dieses Kapitel beinhaltet die Dokumentation für das Kultspiel Snake, angepasst an die Ausführung auf dem 30x30 LED-Board. Das Spiel ist in C++ geschrieben. Bei der Implementierung für unser Projekt ist das klassische Ein-Spieler Spiel zu einem Zwei-Spieler Spiel abgeändert worden. Ziel des Spiels ist es die Schlangen zu füttern und so groß wie möglich werden zu lassen ohne ein Gameover zu erhalten.

5.4.2.1 Spielregeln und Spielende

Die Regeln des Spielesklassikers aus den 1980er Jahren sind weitestgehend gleich geblieben. Die Schlange muss mit den Richtungstasten der Spielsteuerung zu ihrem Futter gelenkt werden. Dabei kann die entgegengesetzte Richtung der Schlangenbewegung nicht benutzt werden, da sie sonst durch sich selbst laufen würde. Ist eine Richtung eingeschlagen, so wird diese weitergeführt, bis eine neue Richtung angegeben oder der Rand erreicht wird. Trifft der Schlangenkopf auf ein Futterobjekt, so verschwindet das Objekt und die Schlange wurde gefüttert. Daraufhin wird die Schlange um eine Einheit verlängert.

Um das Spiel interessanter und aufregender zu gestalten, werden nun zwei Schlangen gleichzeitig auf dem Feld von den Spielern gesteuert. Die Regeln betreffen beide Schlangen gleichermaßen. Die Schlangen sind unterschiedlich gefärbt und können somit auseinander gehalten werden. Jede Schlange muss dem Futterobjekt, welches der eigenen Färbung entspricht. Wird ein gegnerisches Futterobjekt getroffen, verlängert sich der Schlangenkörper nicht und das Objekt bleibt erhalten.

Berührt eine der Schlangen den Rand des Spielfeldes, so endet das Spiel für alle Spieler. Ebenso wird das Spiel beendet, wenn eine Schlange mit dem Kopf den eigenen oder den Schlangenkörper des Gegenspielers trifft.

5.4.2.2 Implementierung

Die Datenstrukturen der Implementierung bestehen aus zwei Klassen „LED Point“ und „Snake“. Das LED-Board ist als eine 30x30 Matrix interpretiert und Objekte der Klasse LED Point befinden sich auf dem Feld. Die Klasse besitzt zwei signed Integer X und Y, welche die Koordinaten des Punktes festlegen. Es handelt sich hierbei um signed Integerwerte, da die untere Grenze des Boards eine -1 ist. Es können zwei Varianten des Konstruktors verwendet werden, entweder mit oder ohne direkter Zuweisung der Koordinaten. Für die Koordinaten sind jeweils Setter und Getter Funktionen vorhanden. Die X und Y-Werte werden mit den move-Funktionen verändert. Die Funktionen legen im Späteren die Richtung der Schlange fest, indem sie die Koordinaten in- oder dekrementieren. Zu Letzt noch die Funktion „copybody()“, welche wichtig für das Nachrücken der einzelnen Körperteile der Schlange ist. Hier werden Koordinaten mit denen im Funktionsaufruf gleichgesetzt.

Die Objekte der Klasse Snake sind ebenfalls LED Points. Die Schlange ist als Array von LED Pointern realisiert, dabei ist die maximale Größe der Schlange mit 50 Elementen gedeckelt. Die Schlange besitzt ein Character-Datentyp für die Richtung, und ein unsigned Integer für die Größe „size“. Zusätzlich verfügt die Klasse über einen boolean

Typ für den Status der Schlange „alive“. Ist der Wert der Boolean-Variable Falsch, so wird das Spiel beendet. Der Konstruktor der Objekte erstellt einen neuen LED Point und setzt alle weiteren Körperteil-Pointer auf NULL. Die Richtung der Schlange wird über die Steuerung mit den turn-Funktionen umgesetzt. Die Klasse verfügt über eine Funktion zur Erkennung von Kollisionen mit entweder dem Rand oder den Schlangenkörperteilen. Liefert die Funktion den Boolean True zurück, so setzt es den Alive-Wert der Schlange auf False und beendet somit das Spiel. Hinzu kommt ein LED Point „food“, welcher zu jeder Schlange ein eigenes Futter LED darstellt.

Soviel zu den Datenstrukturen. Der Beginn der Mainfunktion erstellt die erforderlichen Objekte, Schlangen und Futter. Die Wände werden ausgegeben und den Spielern der Lobby eine Variable zugeordnet. Die Koordinaten der Futterobjekte werden randomisiert, mit der Funktion „set-position()“ gesetzt und danach mit der Funktion „setLED-Point()“ ausgegeben. Die While-Schleife, welche solange durchlaufen wird bis die Lobby geschlossen wird, stellt den Haupttask dar. Bei jedem Durchlauf wird erst überprüft ob die Schlangen der Spieler noch am Leben sind. Ist es eine der beiden nicht mehr, so wird ein Symbol am rechten oberen Rand des Boards ausgegeben und die Spieler der Reihe nach aus der Lobby entfernt. Das Spiel wird beendet und wieder in Main des Haupttasks zurückgekehrt.

Die beiden Spieler können mit der Spielsteuerung nun die Richtungen Oben, Unten, Links und Rechts vorgeben. Die Turn-Funktionen setzen den gewünschten Char-Wert für die gewählte Richtung. Die Schlangen werden in der Funktion „move()“ bewegt. Gefolgt von der Funktion für das Setzen der LEDs auf dem Board und einem notwendigen Hardware Delay für die LEDs. Falls die Lobby unterdessen geschlossen werden sollte, so wird die LED-Ausgabe gelöscht und wieder zum Main-Task des Microkontrollers zurückgekehrt.

5.4.2.3 Die wichtigsten Funktionen

Dieser Abschnitt beschreibt elementare Funktionen und deren Zusammenspiel, welcher einen reibungslosen Ablauf und eine angenehme Spielerfahrung ermöglichen.

- Move():

Diese Funktion ist für den gesamten Bewegungsapparat der Schlangen zuständig. In einem Switch-Case über der Richtung werden die move-Funktionen der Klasse LEDPoint aufgerufen, welche den Koordinaten-/LEDpunkt des Kopfelementes der Schlange bestimmen.

Die Bewegung der Schlange wird wie folgt umgesetzt. Jeder Funktionsaufruf von move() führt die Bewegung des Schlangenkörpers um ein Element in die gesetzte Richtung weiter. Der LED-Punkt des Kopfes wird zunächst unsichtbar gemacht. Eine For-Schleife über dem Schlangenkörper setzt dann für alle Glieder, die nicht NULL sind, die Koordinaten des vorherigen Gliedes mit der Funktion „copybody()“. Auf diese Weise folgen alle Körperteile dem Vorgängerglied bis hin zum Kopf.

Nach dem Setzen der Richtung werden mehrere Werte abgefragt. Zunächst, ob einer der Schlangenköpfe mit seiner nächsten Bewegung die Grenzen des Boards

trifft. Ist dies der Fall, wird der Alive-Wert der Schlange auf False gesetzt und das Spiel beendet. Hiernach folgt die Abfrage, ob die Koordinaten der Schlangenköpfe mit denen der Futterkoordinaten übereinstimmen. Ist der Vergleich wahr, so wird das zugehörige FutterLED gelöscht und ein Körperteil für die Schlange mit der Funktion „appendbody()“ hinzugefügt. Danach werden erneut neue Futterobjekt erzeugt, gesetzt und dargestellt.

Zum Schluss werden alle Körperteile der Schlangen durchlaufen und auf dem LED-Board ausgegeben.

- Check GameOver():
Die Funktion zu Beginn der While-Schleife stellt sicher, ob beide Schlangen noch lebendig sind. Die Funktion „check collision()“ stellt jeweils fest, ob sich die Schlange mit der nächsten Bewegung selbst trifft. Zusätzlich werden zwei Zeiger auf die Köpfe der Schlangen benötigt um weitere Koordinatenvergleiche zu ermöglichen. Für beide Schlangen wird überprüft, ob sie mit der jeweils anderen Schlange kollidieren. Liefert eine der Abfragen ein Boolean True zurück, so wird Alive-Wert aktualisiert und das Spiel beendet.
- SetLED Point: In dieser Funktion werden für die gegebenen LED-Koordinaten der Schlangen und Futterobjekte die Farbwerte gesetzt. Schlangen und ihre eigenen Futterobjekte besitzen die Gleiche Farbe. Das Ansteuern der LEDs ist in dem Kapitel 5.1 konkretisiert.

6 Aufwandsdokumentation

6.1 Reilly Ertman

1. Aufgabe - Stunden in [h]
2. Aufgabe - Stunden in [h]
3. Gesamt = xy

6.2 Khan Ali Muttaqy

1. Aufgabe - Stunden in [h]
2. Aufgabe - Stunden in [h]
3. Gesamt = xy

6.3 Lukas Landgraf

1. Aufgabe - Stunden in [h]
2. Aufgabe - Stunden in [h]
3. Gesamt = xy

6.4 Lisa-Marie Dengler

1. Frontend - 100h
2. Snake - 50h
3. Projektbericht - 20h
4. Gesamt 170h (gerundet, Organisation nicht mitgerechnet)

7 Fazit

Write text here

8 Quellen

- [1] Espressif Systems. *ESP-IDF Programming Guide API Reference*. 13.07.2023. URL: <https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/index.html>.
- [2] Espressif Systems Fromen WiFi. *ESSPRESSIF ESP-IDF Forums post IPC Interfering with RMT*. 13.07.2023. URL: <https://esp32.com/viewtopic.php?f=13&t=24917>.
- [3] ESSPRESSIF Systems. *ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet*. 2023.
- [4] Daniel Weiner. *Video Game Buttons*. URL: <https://codepen.io/DanielWeiner/pen/naybVd>. (besucht: 13.04.2023).
- [5] Co. TM & ©Othello und Megahouse. *eOthello*. 13.07.2023. URL: <https://www.eothello.com/>.
- [6] Prof. Dr. Carsten Kern. *Client-K.I.s für Brettspiele (ReversiXT) Kurzspezifikation*. 2022.

Abbildungsverzeichnis

1	Foto des Boards	2
2	Darstellung der unterschiedlichen Blöcke und deren Schnittstelleninteraktion in unserer Implementierung.	6
3	WS2812b liest eingehend Spannungen bzw. Wörter an der Leitung.	7
4	Ein ws2812b Wort besteht aus einem high gefolgt von einem low Part.	8
5	WS2812b Die Länge eines Wortes in Nanosekunden.	8
6	WS2812b reads timed high and low voltages and 1s and 0s.	9
7	Ein von Flackern betroffener Frame des Spiels Snake	11
8	Die Daten, die auf einen HTTP GET Request auf die URL /Data” als Antwort zurückgesendet werden.	14
9	Liste der Befehle, die über die Websocket für die Lobbyerstellung, Lobbybeitreten und Knopfsteuerung gesendet werden können.	15
10	Das Diagramm zeigt die Kommunikation über die Lobbyklasse zwischen dem Spieldaten und dem Webserver.	16
11	Flussdiagramm zur Navigation durch die Webseite	19
12	Ansicht der Startseite „index.html“ der Webanwendung	20
13	Ansicht der Page „main.html“ der Webanwendung	21
14	Inhalt des lokalen Speichers der Webanwendung	21
15	Ansicht der Page „gameselect.html“ der Webanwendung	22
16	Host-Ansicht(links) und die Spielsteuerung(rechts) eingebettet in die „lobby.html“	23
17	Das linke Bild zeigt ein Othellospiel in der Startaufstellung. Im rechten Bild sieht man dasselbe Spiel, nachdem der grüne Spieler einen Stein rechts neben den oberen rechten Stein des blauen Spielers gesetzt hat.	27

18	Auf dem linken Bild befindet sich das Spielfeld in der Mitte des LED Boards. Der Cursor wurde auf dem rechten Bild nach oben bewegt, aber bleibt konstant in der Mitte des Boards stehen. Stattdessen wurde die Karte nach unten verschoben.	28
19	Das linke Bild zeigt die Auswahl zwischen einem Überschreibstein (O) und einer Bombe (B), das rechte die eines Wahlfeldes. Bei beiden leuchtet der große und kleine Rahmen in der Farbe des Spielers, der die Entscheidung trifft. Der kleine Rahmen markiert, was gerade ausgewählt ist.	29
20	Dieses Bild wird vor jedem Spielzug in der Farbe und mit den Daten des Spielers, der nun am Zug ist, angezeigt. In diesem Fall handelt es sich um Spieler 3 mit vier Überschreibsteinen (O) und einer Bombe (B).	29
21	Kommandozeilenausgabe der 3 Spieler Karte aus Abbildung 18 für Debugging.	30
22	Die Farben, die für das LED Board verwendet, und die ASCII Zeichen, die für die Kartenerstellung und serielle Schnittstelleausgabe genutzt werden.	31

Abbildung 3: <https://www.arrow.com/en/research-and-events/articles/protocol-for-the-ws2812b-programmable-led> Abbildung 4+5+6: Direkt aus LED Datablatt