Digital Detectives

Ein Projekt von Noah Mühl und Philipp Hinz

20.12.2019

Inhaltsverzeichnis

Vielen Dank an Lasse Steglich, der uns bei elektrotechnischen Fragen beraten hat.

0. Vorwort

Das Ziel dieser Dokumentation ist es das Spielen so leicht wie möglich zu gestalten, für Interessierte einen Einblick in die Digitalisierung eines Brettspiels zu geben und für Entwickler ein gutes Fundament und Nachschlagewerk zur Weiterentwicklung zu bieten. Es wurde versucht diese Dokumentation umfangreich genug zu gestalten, dass man auch mit wenigen technischen Kenntnissen in der Lage ist, ein solches oder speziell dieses Projekt Nachzubauen.

1. Projektskizze

In diesem Kapitel geht es um die Vorplanung und die Stufen, welche unser Projekt durchläuft oder durchlaufen hat.

1.1. Konzept

Unser Konzept des Projektes ist die Digitalisierung des Brettspiels Scotland Yard. Dies ist ein Brettspiel für zwei bis sechs Spieler, welches in der Regel 60 Minuten dauert. Dabei gibt es Detektive, welche einen Bösen jagen. Dieser muss versuchen in bis zu Insgesamt 24 Runden vor diesen zu fliehen. Dabei darf er nicht erwischt werden.
Das Spiel soll mit Hilfe eines Brettes mit RGB-LEDs dargestellt werden, eine Anzeige zur Angabe von simplen Informationen darstellen und eine Website bereitstellen, welche den Spielern die Möglichkeit gibt, Züge auszuführen.

1.2. L1 - Vorbereitung

In dieser Stufe wollen wir das Projekt planen und grundlegende Strukturen sowie Modelle aufbauen. Dabei definieren wir nötige Anforderungen für das Projekt und fangen an eine Architektur zu entwickeln, welche diese erfüllt. Diese Architektur soll softwaretechnisch umgesetzt und getestet werden. Falls Anforderungen in geplanter Weise nicht erfüllt werden können, werden diese neu geplant. Dieser Anfang kann auch als Proof-of-concept betrachtet werden. Hier nochmal die wichtigsten Punkte.

- Planung
- Architektur
- Testen
- Aufbauen

1.3. L2 - Umsetzung

Nun wollen wir die vorher geplante Vorbereitung umsetzen. Die Hardware soll funktionsfähig aufgebaut werden, jedoch noch nicht dem finalen Produkt nahekommen, womit gemeint ist, dass nicht das Äußere wichtig ist, sondern nur die Funktion an sich erreicht werden soll. Die Software soll im Ganzen der Architektur nach entwickelt werden und funktionsfähig sein.

Dabei soll zunächst auf eine Kommunikation über die serielle Schnittstelle gesetzt werden, da ein Zugriff über eine Website als sehr kompliziert angesehen wird. Im Ganzen soll am Ende dieser Phase das Spiel theoretisch funktionsfähig sein, jedoch in keinem Fall benutzerfreundlich oder schön sein.

1.4. L3 - Konstruktion

Bei der Konstruktion geht es darum, aus den Komponenten ein fertiges Spiel zu fertigen. Es soll benutzerfreundlich durch ein Webinterface sein, solide und schön aussehen, sowie in angeforderter Weise funktionieren. Das heißt, dass das Web-Interface implementiert werden muss und mit dem Spiel interagiert. Das Brett muss als festes Gehäuse dienen, welches das Spiel durch ein verständliches Spielfeld darstellt. Ein Spiel soll möglichst ungestört in korrekter Weise spielbar sein.

1.5. L4 - Konzepte

Als weiteres sind Konzepte geplant, welche das fertige Spiel erweitern sollen. Dabei wären die beiden primären Konzepte, künstliche Intelligenz und eine weitere Zugriffsmöglichkeit zu dem Web-Interface. Diese würde Spielern eine Möglichkeit zum Spielen geben, welche keinen Zugriff auf ein HTTP fähiges Gerät verfügen. Dieses wäre durch ein weiteres Gerät auf der Basis eines ESP32 bereitgestellt. Die künstliche Intelligenz sollte dem Nutzer ermöglichen, bei zu wenigen Spielern eine KI stattdessen spielen zu lassen. Außerdem wäre es möglich einen Spieler, welcher das Spiel verlässt, zu ersetzen.

2. Hardware

In diesem Abschnitt wird die hardwaretechnische Seite des Projektes näher erläutert. Ferner wird dabei auf die verwendeten Komponenten und deren Funktionsweise eingegangen sowie die Verkabelung. Auf die konkrete Implementierung der Hardware und ihre Verschaltung wird im Technologieschema näher eingegangen.

2.1. Mikrocontroller

Für die Steuerung und Verwaltung des Brettes und der restlichen Hardware nutzen wir einen leistungsstarken Mikrocontroller, den ESP32 der Firma expressif und dem Hersteller AZ-Delivery mit zwei Kernen, über 1 MB Flashspeicher und mehr als 500 KB Ram. Wir haben uns für diesen entschieden, da er eine vergleichsweise enorme Leistung bietet, von Anfang an aus WLAN mit WPA2 Verschlüsselung bietet sowie weit verbreitet ist. Außerdem sind alle nötigen Bibliotheken vorhanden und ist mit Hilfe der Arduino Entwicklungsumgebung programmierbar.

2.2. Brett

Das Gehäuse unseres Spieles besteht aus einer Holzbasis, welche mit einem Boden aus Holz und einem Dach aus Plexiglas und Messing ausgestattet ist und vollkommen mit einem

Aluminiumrahmen aus einem U-Profil umschlossen wurde. Dabei besteht der Großteil des Daches aus Plexiglas und nur ein kleiner Bereich für die Steuerung wird mit Messing bedeckt. Innerhalb des Brettes sollen die Steuerung und Verkabelung platziert werden. Wichtig ist zudem, dass der Boden im Gegensatz zu den anderen Komponenten nicht festgeklebt, sondern nach Belieben verschiebbar ist.

2.2.1. Plexiglas Dach

Das Plexiglas-Dach dient als Spielfeld. Es wurde von unten mit RGB-LEDs beklebt und von oben schwarz lackiert. Alle LEDs wurden miteinander wie im Punkt 2.3. WS2812B angegeben, verlötet und das Spielfeld wie im Originalspiel Scotland Yard bemalt. Dabei wurde das Plexiglas im Rahmen festgeklebt.

2.2.2. Messing Dach

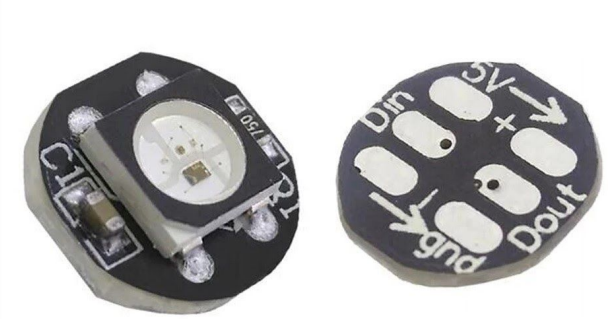
Das Messing-Stück wurde für die Steuerung beschnitten und enthält ein LCD, drei Knöpfe und eine weitere Anzeige für das Spiel. Diese Anzeige besteht aus eine von unten geklebte Plexiglasscheibe, welche ähnlich wie das Plexiglasdach lackiert, gelötet und bemalt wurde. Das Messing ist im Vergleich des restlichen Brettes sehr massiv und leitet, weswegen man sich vor Kurzschlüssen achten musste.

2.2.3. Boden

Der Boden besteht aus Holz und ist nicht an das restliche Brett geklebt. Der Boden lässt sich mit einer Kante des Rahmens rausziehen, welcher ebenfalls nicht an das restliche Brett geklebt ist. Der Boden ist außerdem grau lackiert.

2.3. WS2182B

Da unser Spiel über 200 bunte LEDs nutzt und wir nur sehr wenige Pins zur Verfügung haben, haben wir uns für indexierbare LEDs entschieden. Diese erlauben eine unbegrenzt lange Schlange an LEDs, welche in Reihe verbunden wird. Dabei werden diese durch eine Bibliothek mittels einer Bus-Verbindung gesteuert.



Wir haben uns speziell für das Modell WS2182B entschieden, da dieses uns von der Preisleistung am effizientesten erschien. Wir haben insgesamt 300 LEDs bestellt, da diese primär beim Testen und Aufbauen schnell kaputt gehen.

2.4. Knöpfe

Für die Knöpfe nutzen wir spezielle für Vandalismus geschützte Knöpfe, da diese einen sehr kleinen Rand oben lassen. Dadurch kann das Brett auch auf den Kopf gelegt werden, ohne dass die Gefahr vor einer Beschädigung des Knopfes besteht.

2.5. LCD

Für eine simple Kommunikation mit dem Nutzer nutzen wir ein LCD. Dabei haben wir uns für eine größere Variante entschieden, nämlich einen der Größe 20x4. Außerdem nutzen wir den Adapter I2C für das LCD, damit dieser nur zwei Pins verwendet. Dabei kommuniziert der I2C mit seriellem Protokoll.

2.6. Stromversorgung

Die Stromversorgung wird mittels vier 3.6v Lithium-Ionen-Akkus gewährleistet. Dabei werden diese durch einen Battery-Charger-Manager verwaltet. Der ESP32 liegt normalerweise nicht an der Stromversorgung, sondern kann mit dem mittleren Knopf an diese verbunden werden. Die insgesamte Kapazität der Stromversorgung beträgt 10Ah, welche mindestens acht Stunden Spielzeit bietet.

2.7. Schaltkreis zur Stromversorgung

In dem Schaltkreis für die Stromversorgung wurden mehrere Widerstände und ein Mosfet genutzt. Auf diese wird später näher eingegangen.

2.8. Komponenten

Im Detail haben wir folgende Komponenten genutzt:

- ESP32 von AZ-Delivery
- LCD 20x4 mt I2C
- Mosfet FQP30N06L
- Widerstände 2x 22 K, 1x 1.2 K, 1x 470, 3x 1K
- Aluminium U-Profil 3m bei 10mm x 35mm x 10mm
- Plexiglas 65 cm x 40 cm x 2 mm
- Plexiglas 17 cm x 8 cm x 1 mm
- Messing 35 cm x 40 cm x 2 mm
- Holzplatte 80 cm x 40 cm x 27 mm
- Holzplatte 80 cm x 40 cm x 6 mm
- WS2182B x 223
- Kupferkabel “Litze” isoliert 0.12mm² x 20 m
- Vandalismus geschützte Knöpfe x 3 mit 12 mm²
- RGB LED x 1
- Acrylfarbstifte Rot, Grün, Weiß und Gelb
- Acrylfarblack Schwarz, Grau, Klar

- Akku US18650VTC5 3.6V 2600mAh x 4

3. Technologieschema

In diesem Abschnitt wird nun auf die Implementierung und Verkabelung der Hardware eingegangen. Dabei gehen wir zuerst auf die theoretischen Anforderungen der Hardware ein und behandeln danach die praktische Umsetzung.

3.1. Hardwaretechnische Analyse

Bevor wir uns die Funktionsweise und den Aufbau unserer Hardware anschauen, besprechen wir noch kurz die Anforderungen, welche an die hardwaretechnische Seite gestellt werden. Dabei definieren wir die Anforderungen der Hardware als externe Anforderungen. Dazu im Gegensatz stehen die internen Anforderungen, auf welche in der Softwareverteilung näher eingegangen wird.
Unsere Hardware versucht das Spiel in bester Weise darzustellen, wobei alle Elemente intuitiv, interaktiv und automatisch gestaltet werden sollen, da wir versuchen den Schwerpunkt des Projektes beim eigentlichen Spielen, zu lassen und versuchen zu verhindern, dass die Spieler nur auf ihre Handys schauen. Dies erreichen wir indem wir nur Informationen auf den Handys darstellen, welche nur diese Person wissen darf und sonst alle anderen Informationen auf der Hardware darstellen. Komponenten wie das Spielfeld sollen nicht auf der Website, sondern nur durch die Hardware dargestellt werden. Generell muss das Brett durch bunte LEDs alle Spieler darstellen und die Pfade mit Bunter Farbe, welche aufgemalt werden müssen. Tickets, welche der Böse genutzt hat, müssen ebenfalls durch bunte LEDs angezeigt werden und ein LCD, welches simple Informationen anzeigt, welche wichtig für alle Nutzer sind. Über drei Knöpfe soll es die Möglichkeit geben auf Menüs zugreifen, welche Einstellungen am und das Starten des Spiels ermöglichen. Als weitere Anforderung setzen wir uns, das Einschalten durch einen Knopf. Dabei soll die Stromversorgung im Normalfall völlig ausgeschaltet sein und nur durch das Drücken des Knopfes ausgeschaltet werden. Dabei wird dieser Knopf zum Anschalten als normaler Knopf wiederverwendet und soll das Gerät nicht wieder Ausschalten. Das Ausschalten hingegen soll Softwaretechnisch durch den ESP32 gehandhabt werden, indem er einen digitalen Pin setzt.

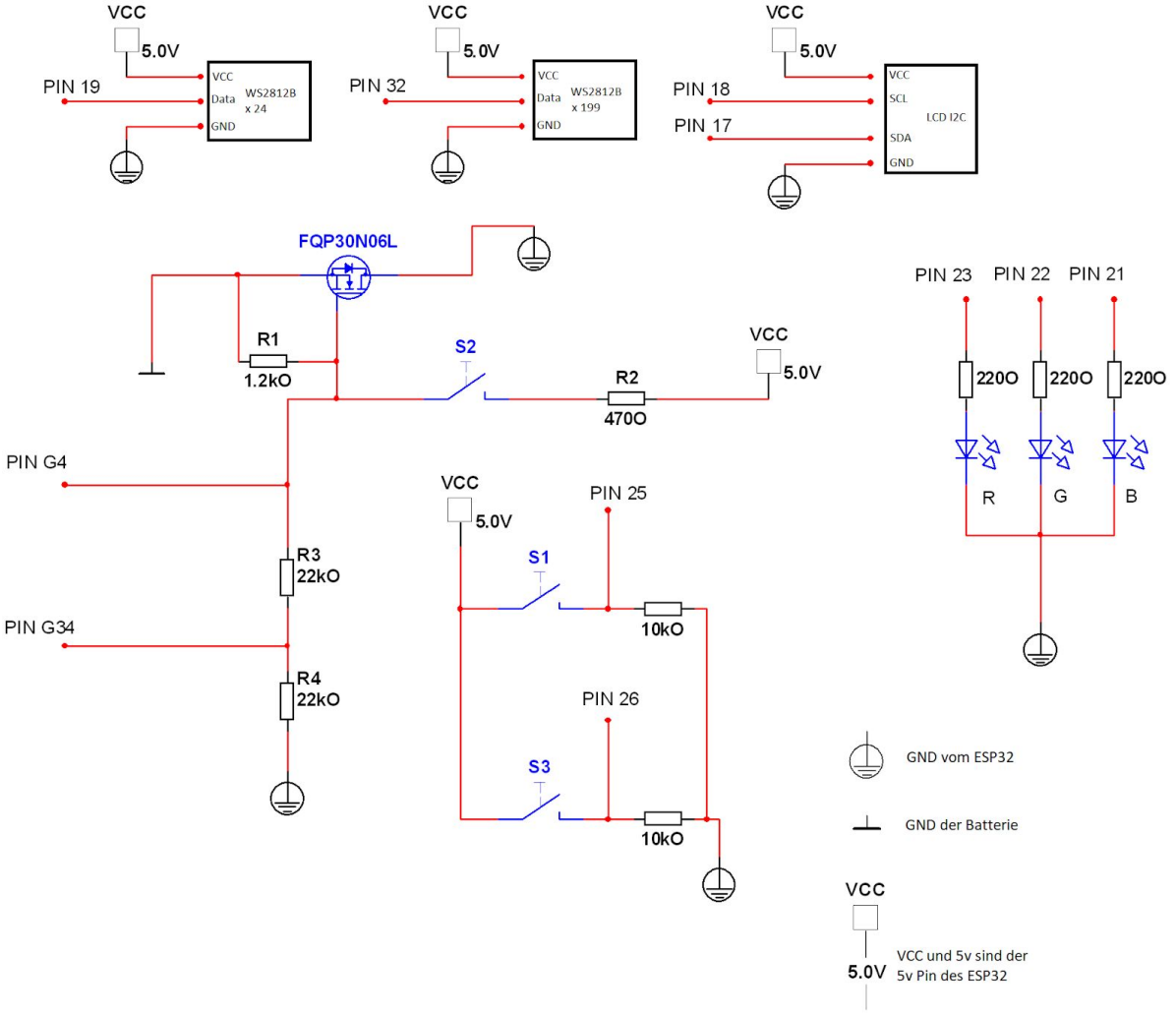
3.2. Umsetzung

Nun schauen wir uns die konkrete Umsetzung der Hardware an und gehen zu Beginn auf den Schaltkreis ein und schauen uns danach noch den Bauplan an, wobei der Bauplan das Gehäuse darstellt. Wie Schaltkreise konkret verbaut wurden ist nur auf Bildern zu sehen.

3.2.1. Schaltkreis

In der folgenden Abbildung ist der Schaltkreis zu sehen. Alle drei Knöpfe sind ohne Pullup oder Pulldown Widerstand angeschlossen, da welche intern vom ESP32 genutzt werden. An der RGB LED werden drei 1K Widerstände genutzt, welche die LED vor Überlastung schützen. Die WS2182B LEDs sind direkt angeschlossen, da diese eine 5V Verbindung

brauchen und jeweils einen internen Verwaltungs-IC besitzen. Der LCD wurde an einen I2C geschlossen, da dieser die Anzahl an nötigen Pins massiv erniedrigt (mit I2C nur zwei).

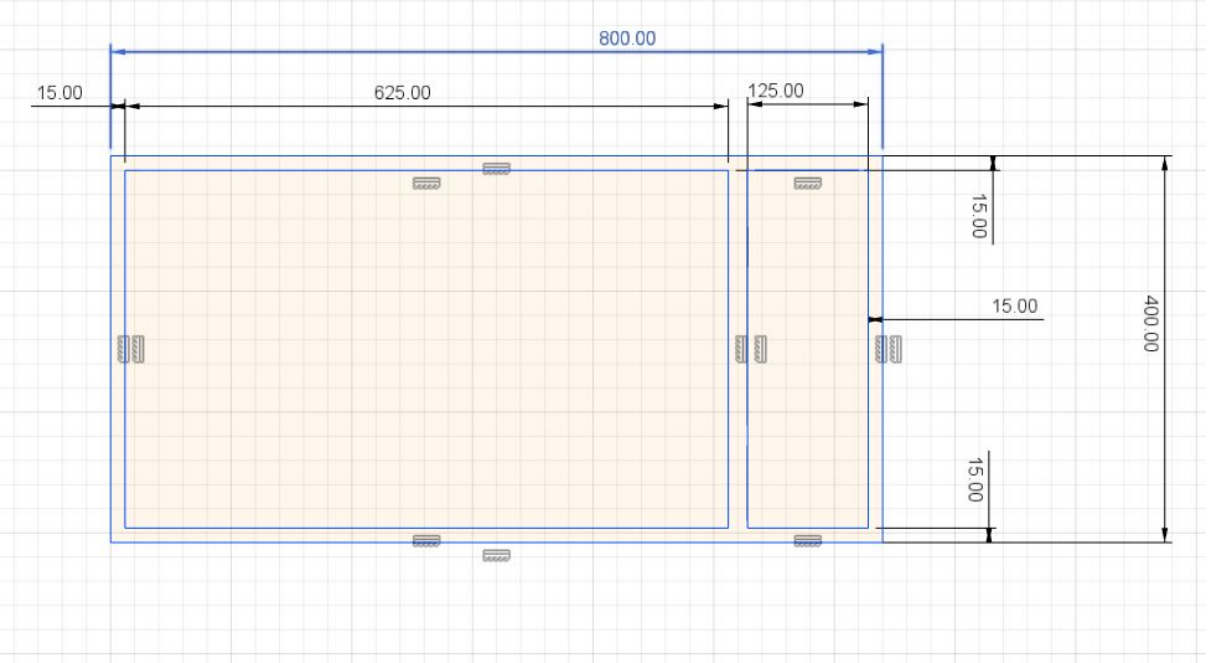


3.2.2. Bauplan

Im Folgenden wird auf den Bauplan und die Umsetzung im praktischen eingegangen. Dabei wird anhand von Modellen und Bildern sowie Maßen versucht ein verständliches sowie akkurates Bild des Spiels zu vermitteln.

3.2.2.1. Gehäuse

In der folgenden Abbildung ist der Bauplan des Gehäuses zu sehen, welches im CAD modelliert wurde. Das Gehäuse besteht aus einer Holzbasis mit der Länge 85 cm x 40 cm, welches innen hohl ist. Es bleibt ein 1.5 cm breiter Rand übrig, wobei nach 64 cm ein Zwischenstück mit der Länge von 2 cm auftritt. Der rechte Ausschnitt dient als Halterung für das Spielfeld und das linke als Halterung für das Messing.

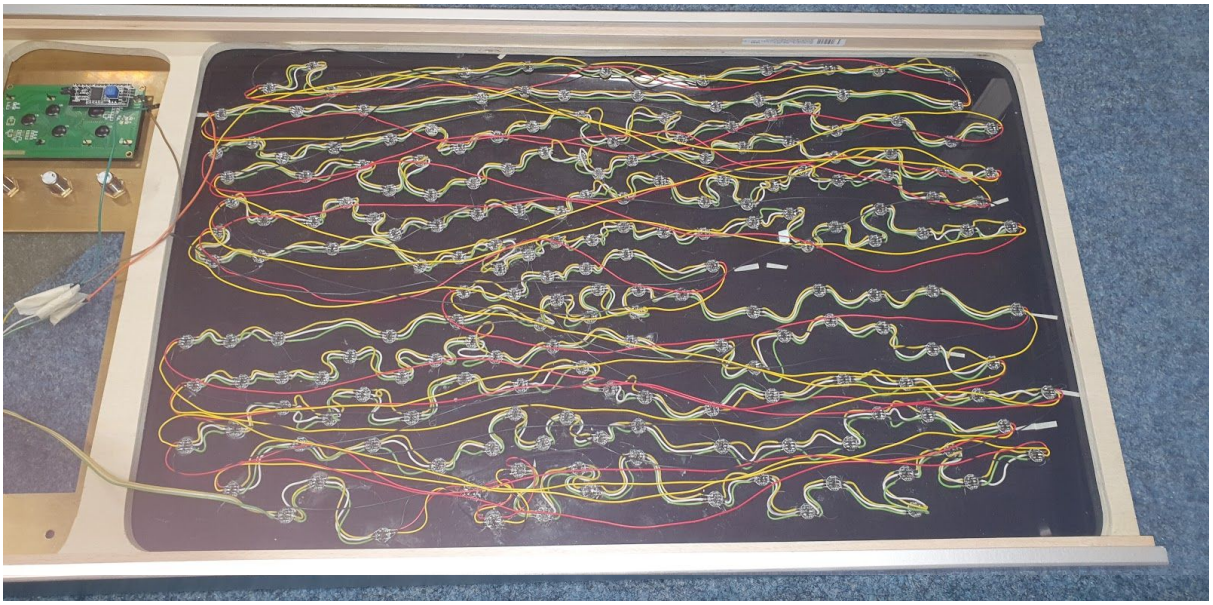


3.2.2.2. Darstellung Spielfeld

Auf dem Plexiglas mit den Maßen 65 cm x 40 cm x 2 cm wurden alle Felder abgeklebt und schwarz lackiert. Darauf wurde dann mit Acrylfarbe das Spielfeld aufgemalt, welches in der folgenden Abbildung gezeigt wird. Zum Schluss ist das Spielfeld nochmal mit Klarlack überdeckt, damit es Widerstandsfähig ist.

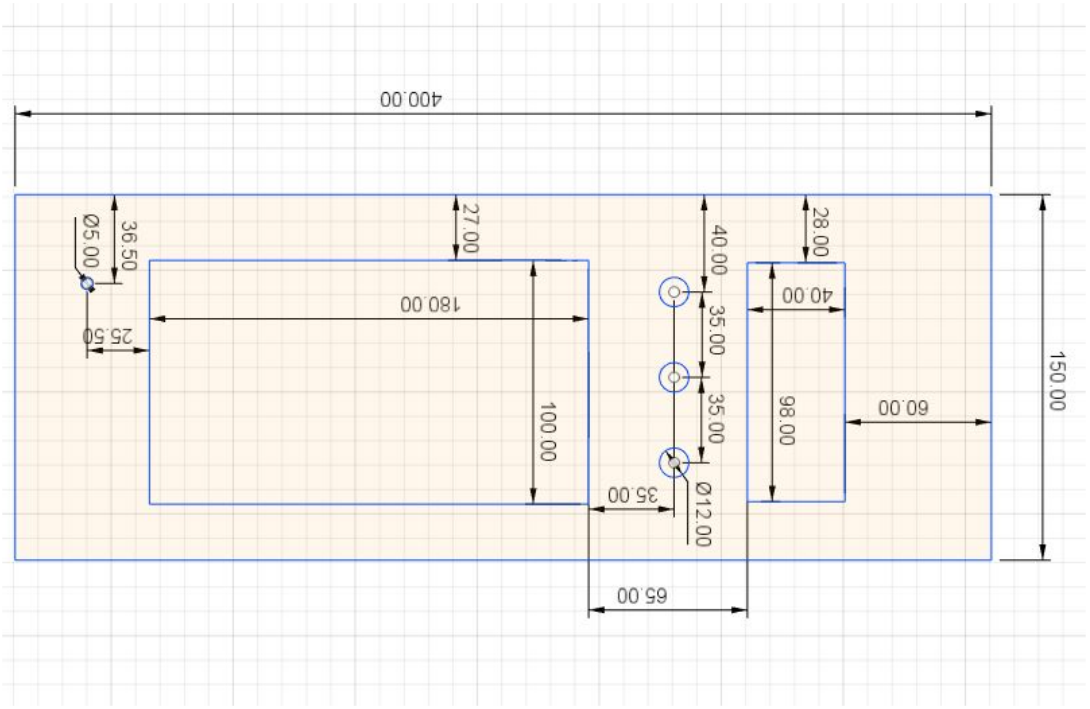
3.2.2.3. Verkabelung Spielfeld

Die 199 x WS2192B LEDs wurden mit Hilfe von Heißkleber an dem Spielfeld (Plexiglas) festgeklebt und dann mit Lötzinn verlötet. Dabei wurden sie mithilfe eines 0.12 mm² Litze Kupferkabel verbunden, woraus sich insgesamt um die 1200 Lötstellen und 600 Kabel ergaben. In folgender Abbildung ist diese Verkabelung zu sehen.



3.2.2.4. Informationsfeld

Das Informationsfeld besteht aus Messing mit den Maßen 25 cm x 40 cm x 2 cm und hat zwei viereckige Ausschnitte mit den Maßen 4 cm x 9.8 cm sowie 18 cm x 10 cm und vier runde Löcher mit den Maßen 12mm² sowie 5 mm². Dabei dienen die Ausschnitte für das LCD und die Informationsanzeige für die Detektive. Drei der Löcher gehören zu den Knöpfen und eines der Status LED. Im folgenden Bild ist dieser Ausschnitt zu sehen.



4. Softwareverteilung

In diesem Abschnitt betrachten wir die Softwaretechnische Seite unseres Projektes. Dabei gehen wir zuerst auf die Ziele und Anforderungen der Software ein, legen grundlegende Konventionen fest und dokumentieren sowie erklären alle existierenden Module sowie deren Abhängigkeiten. Zusätzlich erklären wir am Ende alle Fehlermeldungen und deren Lösungen.

4.1. Ziele der projektspezifischen Softwareentwicklung

Unser Projekt besitzt die externe Anforderung zur Definierung eines abstrakten softwaretechnischen Konstrukts zur Repräsentation des Spieles über die Hardware und der Definierung eines für den Nutzer einfachen, sowie freundlichen Zugangspunktes über eine ihm leicht zur Verfügung stehende und zugängliche Schnittstelle. Diese externe Anforderung wird durch eine interne erweitert, durch welche das Projekt definierte softwaretechnische Qualitätsattribute erfüllen soll, auf welche gleich weiter eingegangen wird.

4.1.1. Externe Anforderungen

Da die externen Anforderungen bereits in der Projektskizze sowie im Technologieschema näher erläutert wurden, wird hier nur auf die direkten softwarespezifischen Aspekte näher eingegangen.
Da dieses Projekt auf Basis eines Mikrocontrollers sowie der Arduino Entwicklungsumgebung basiert, ist diese Software von Grund aus sehr fehleranfällig. Durch die zusätzliche Belastung durch mögliche fehlerhafte Hardwarekomponenten sehen wir uns gezwungen eine komplexe, sowie mehrstufige Fehlerbehebung einzubauen, auf welche später näher eingegangen wird. Da auf Arduino basierende Mikrocontroller wie der unserer

eine sehr limitierte Speicherkapazität besitzen, verzichten wir Größtmöglich auf dynamischen Speicher sowie Objektorientierung.

4.1.2. Interne Anforderungen

Vor der Entwicklung unseres Projekts mussten wir uns für eine Entwicklungsstruktur entscheiden und haben uns für die architekturbasierte Entwicklung entschieden. Primär führt dies zu einer stark kontrollierten Entwicklung (nicht als eingeschränkt gemeint), jedoch werden wir aus Gründen der Priorität nicht näher auf die Gründe dieser Entscheidung eingehen.

Die für uns zum Beginn primäre Frage der architekturbasierten Entwicklung, ist die der Qualitätsattribute. Diese geben unserem Projekt feste Prioritäten wie zb. Sicherheit oder Skalierbarkeit und ermöglichen uns vorab einen langfristigen Plan zu erstellen und mögliche Probleme zu verhindern.
Funktionalität wird hier nicht als geltendes Qualitätsattribut betrachtet, sondern als fertiges Ergebnis der Software und Summe aller Qualitätsattribute.

4.1.2.1. Qualitätsattribut Sicherheit

Sicherheit heißt Schutz von sowie Prävention vor externen Angriffsquellen.

Unser Projekt befindet sich in einem lokalen, nicht ökonomisch orientierten sowie meist freundschaftlichen Umfeld, weshalb keine Wichtigkeit in der Sicherheit der Software liegt. Es wird von vorhinein nicht davon ausgegangen, dass dieses Projekt jemals als Ziels eines Angriffes liegen wird, weshalb wir keine zusätzliche Arbeit in die Sicherheit setzen werden und uns auf mehr auf andere Qualitätsattribute konzentrieren.

4.1.2.2. Qualitätsattribut Skalierbarkeit

Skalierbarkeit bedeutet einfach erweiterbar und skalierbar.

Wir haben dieses Qualitätsattribut als eines der Primären festgelegt, da wir eine einfache sowie kontinuierliche Weiterentwicklung der Software ermöglichen wollen. Ein Beispiel dieser wäre die offene Schnittstelle der Kommunikation, welche uns bereits ermöglicht hat, durch einfache Wege Komplexe Bot (KI) Systeme hinzuzufügen. Ein weiterer Punkt wäre die Skalierbarkeit der Hardware, durch welche wir einfach weitere Hardware hinzufügen und managen können. Ein primäres Problem der Skalierbarkeit ist die dadurch erzeugte Komplexität, welche auch die gesamte Größe des Projektes drastisch erhöht.

4.1.2.3. Qualitätsattribut Zuverlässigkeit

Zuverlässigkeit bedeutet Resistenz gegen Störung der Funktionalität. Konkret, Abbruch eines Spieles durch softwaretechnische Fehler.

Dieses Qualitätsattribut ist so ziemlich das wichtigste des ganzen Projektes. Da die ganze Funktionalität alleine darauf basiert, dieses Spiel zu spielen, besonders: es bis zum Ende zu spielen, liegt die Prävention des Abbruchs des Spieles an oberster Stelle. Wir haben große

Umwege, hohe Komplexität und Verzicht an anderen Qualitätsattributen in Kauf genommen, um eine möglichst hohe Zuverlässigkeit der Software zu erreichen.

4.1.2.4. Qualitätsattribut Nutzbarkeit

Nutzbarkeit bedeutet Benutzerfreundlichkeit und Bedienbarkeit.

Da dieses Spiel besonders auch für jüngere Menschen geeignet ist, man bedenke, dass das Spiel, auf welchem dieses hier basiert, ab acht Jahren freigegeben ist, ist die Nutzbarkeit keine unwichtige Komponente. Wir haben größere Hürden in Kauf genommen, genauer gesagt uns für ein Webserver entschieden, damit dieses auf nahezu allen Geräten ohne lästigen Download zugänglich ist. Jedoch haben wir uns dagegen entschieden, diese Website für weitere Einstellungen und ähnliches zu nutzen. Stattdessen wird dafür ein LCD und Knöpfe genutzt.

4.1.2.5. Qualitätsattribut Performance

Performance bedeutet Geschwindigkeit und Ausfallsicherheit (nicht Wiederherstellungsmöglichkeit).

Die Performance unseres Projektes liegt eher an hinterer Stelle, da wie es häufig bei Brettspielen ist, keine Echtzeit-Entscheidungen getroffen werden. Deswegen haben wir an manchen Stellen größere Einbüßen an Performance in Kauf genommen und uns dafür mehr auf andere Qualitätsattribute konzentriert.

4.1.2.6. Qualitätsattribut Modifizierbarkeit

Modifizierbarkeit bedeutet Anpassung von Strukturen, Daten und Funktionen.

Modifizierbarkeit war uns nicht in gleichem Maße wie Skalierbarkeit wichtig, jedoch ist dies auch ein wichtiger Punkt, da eine Anpassung und mögliche Erweiterung (ähnlich wie Skalierung, jedoch in unterschiedlicher Weise) in Zukunft noch wichtig sein könnte. ZB. die Möglichkeit der Änderung der Datentypen bzw. Datenkonstrukte war uns nicht unwichtig. Außerdem sind Anpassungen an der Repräsentation nicht unwichtig.

4.1.2.7. Qualitätsattribut Portierbarkeit

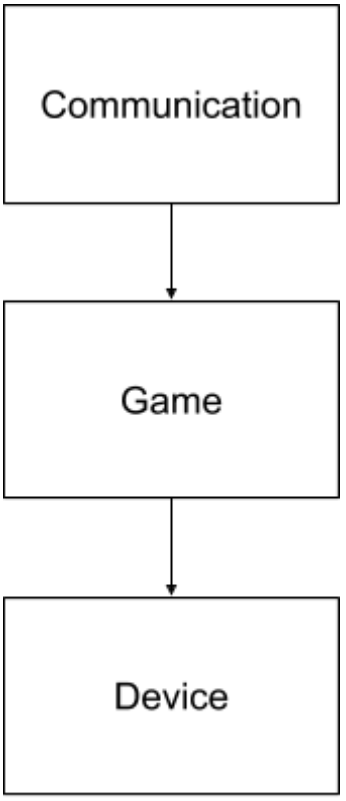
Portierbarkeit bedeutet der mögliche Wechsel von Hardware und Bibliotheken.

Da wir nicht davon ausgehen konnten, dass unsere Hardware für die Digitalisierung des Spieles ausreicht (zurecht; siehe 5.2.3. ESP8266, Probleme). Außerdem war uns wichtig, dass ein Wechsel von Bibliotheken leicht möglich ist, damit auf möglicherweise unzureichende verzichtet werden kann.

4.2. Architektur

Die grundlegende Idee unserer Architektur ist es, die Software in drei Ebenen zu unterteilen, welche ein Interface zur Verfügung stellen, auf welches immer nur die jeweils unteren zugreifen können. Dabei unterscheiden sich die Ebenen in deren Aufgaben und sind

funktional unabhängig voneinander. Das heißt, dass die innere Funktion uneingeschränkt geändert werden kann, solange das Interface der Ebene gleichbleibt. Eine Ausnahme dieser Regel ist das Common Modul, welches Definitionen und Bibliotheken zur Verfügung stellt, die häufig genutzt werden.



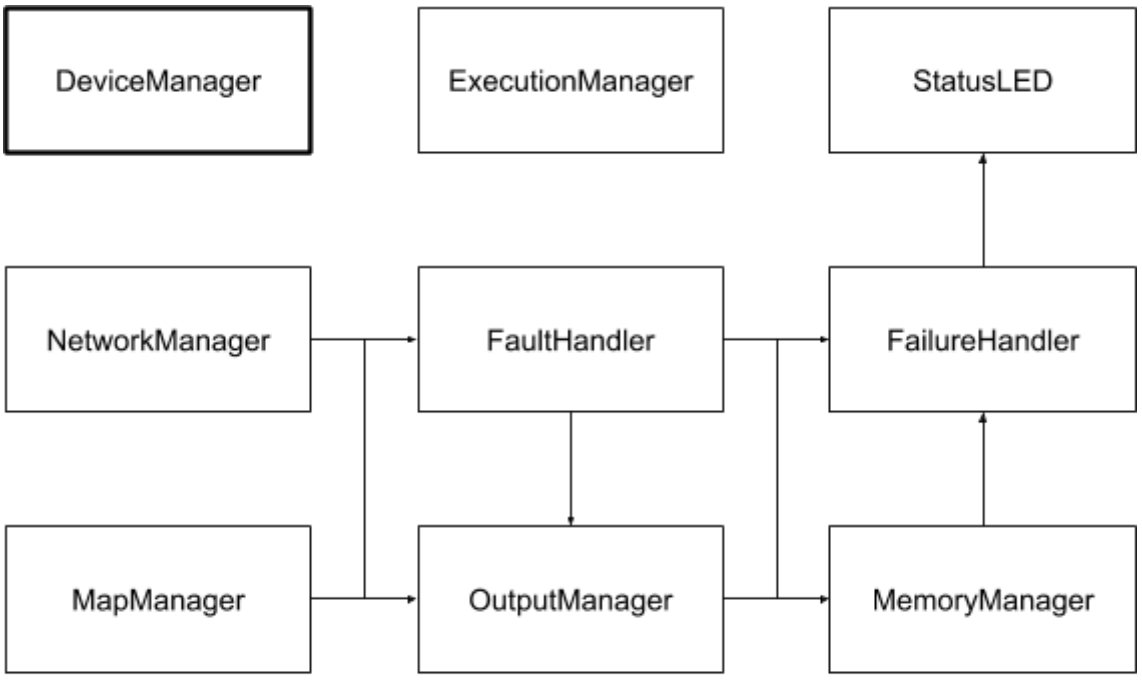
Alle Ebenen

Hier (Bild rechts) werden die drei Ebenen dargestellt. Hierbei dient die Device-Ebene als grundlegende Abstraktion der Hardware, weshalb bei einem Hardwarewechsel nur diese Ebene angepasst werden muss (abgesehen von Bibliotheken). Diese kümmert sich um grundlegend gerätespezifische Aufgaben, wie das Speichermanagement, die Ausgabe durch Hardware oder die Fehlerbehandlung. Die Game-Ebene dient als Simulation des Spiels. Mit dem bereitgestellten Interface können alle nötigen Spielzüge ausgeführt und Daten ausgelesen werden. Bis auf die Prüfung von illegalen Spielzügen haben wir uns dagegen entschieden, die Abfrage der Information einzuschränken, da ein direkter Angriff der Software während eines Spieles ausgeschlossen wird und jeglicher Zugriff auf das Interface bereits von uns gesteuert wird. Wichtig noch: diese Ebene ist intern in mehrere linear ablaufenden Phasen unterteilt, welche einen Spielstatus darstellen (wie z.B. “Spiel läuft” oder “warte auf Spieler”) und über das Interface der Ebene gewechselt werden können. Die oberste Ebene ist Communication, welches wie der Name schon sagt die Kommunikation des Spieles selbst mit dem Nutzer verwaltet. Dabei kann diese Ebene mit beliebig vielen Zugriffsmöglichkeiten erweitert werden, welche der Nutzer zum Spielen nutzen kann. Die primäre Zugriffsmöglichkeit ist das Webinterface. Wichtig noch: Die Aufgabe dieser Ebene ist es den Zugriff auf Informationen einzuschränken.

Im nächsten Abschnitt gehen wir nun näher auf die Ebenen und die einzelnen Module ein. Dabei ist anzumerken, dass mit Modulen keine Klassen gemeint sind. Module haben eine Aufgabe und stellen ein Interface im Sinne von Funktionen zur Verfügung.

4.2.1. Device-Ebene

Die Device-Ebene stellt wie im vorherigen Abschnitt erwähnt, die Abstraktion der Hardware dar und ermöglicht dadurch den anderen Ebenen einen leichten Zugriff auf diese. Diese Ebene besteht aus insgesamt neun Modulen, von welchen der DeviceManager die zentrale Verwaltung des Gerätes darstellt. Diese Module stehen in folgender Abhängigkeit zueinander (siehe folgendes Bild), wobei der DeviceManager Zugriff auf alle Module hat, welcher hier nicht dargestellt wird. Der ExecutionManager hat im Gegensatz weder Zugriff auf Module, noch hat ein Modul Zugriff auf diesen.



Device Ebene

4.2.1.1. DeviceManager

Dieses Modul stellt die Verwaltung des gesamten Systems dar und hat deswegen als einzige Ausnahme Zugriff auf alle Module, auch die anderen Ebenen. Zum Beginn der Ausführung wird direkt aus dem Einstiegspunkt mithilfe dieses Moduls das System initialisiert. Nach der Initialisierung wird das Spiel wieder mittels dieses Moduls ausgeführt. Auf dieses Modul sollte nicht von außen zugegriffen werden, da es keine für andere Ebenen relevante Funktion bereitstellt.

4.2.1.2. ExecutionManager

Der ExecutionManager spielt bisher eine nur sehr untergeordnete Rolle und ermöglicht nur eine Überprüfung des gerade laufenden Prozessorkerns.

4.2.1.3. StatusLed

Dieses Modul kümmert sich hauptsächlich um eine einzige RGB-LED und dient als letzte Möglichkeit, im Sinne eines Fehlers, der Kommunikation.

4.2.1.4. FailureHandler

Der FailureHandler ist der erste der beiden Fehler-Behandler und kommt nur selten bei unbehebbaren Fehlern zum Einsatz. Die einzelnen Fehlercodes werden über die Status-LED ausgegeben, auf welche im Kapitel der Fehlercodes näher eingegangen wird.

4.2.1.5. MemoryManager

Dieses Modul verwaltet den dynamischen Speicher sowie den EEPROM, dabei wird auf dem ESP für den EEPROM der Flash Speicher verwendet. Der EEPROM wird in Sektoren

unterteilt, auf welche andere Module zugreifen können. Alle Sektoren werden automatisch von diesem Modul signiert, heißt sie sind vor fehlerhaftem Schreiben und Lesen geschützt. Außerdem signiert es auch selbstständig den EEPROM und verhindert dadurch einen Fehler beim Schreiben und Lesen. Außerdem bietet dieses Modul die Möglichkeit, wenn nötig dynamischen Speicher anzulegen.

4.2.1.6. OutputManager

Alle Ausgabemöglichkeiten wie das LCD, die LEDs, Knöpfe, usw. werden durch den OutputManager verwaltet. Dabei erlaubt dieses Modul auch eine einfachere Eingabe und Auswertung durch Knöpfe. Außerdem enthält der OutputManager das Untermodul Interact, welches Funktionen zur einfachen Eingabe, sowie eine Funktion zum Starten eines Menüs besitzt. Diese Funktion wird bisher nur von LcdAccess (siehe 4.2.3.1. LcdAccess) genutzt.

4.2.1.7. FaultHandler

Der FaultHandler ist die zweite Stufe der Fehlerbehandlung und hat im Gegensatz zum FailureManager auch Zugriff auf den OutputManager, wodurch es die Möglichkeit hat dem Benutzer die Fehler leichter anzuzeigen. Dabei hat der FaultHandler außerdem noch die Aufgabe während des laufenden Spiels nach Fehlern im Gerät zu suchen, wie zb. ein fehlerhafter oder überlaufender Speicher. Wichtig noch: Als einzige Ausnahme hat der FaultHandler Zugriff auf das Modul InterfaceManager der Communication-Ebene mittels eines Callbacks, um die Anzeige von Fehlern dem Nutzer auch über sein Endgerät zu ermöglichen.

4.2.1.8. MapManager

Die einzige Aufgabe des MapManagers besteht darin, alle LEDs des Spielfeldes zu indexieren und anderen Modulen eine einfache Möglichkeit zu bieten, zwischen Spielfeld-LEDs und echten LEDs zu wechseln. Dies ist nötig, da die Reihenfolge der Felder auf dem Spielfeld nicht die Reihenfolge der tatsächliche Leds entspricht.

4.2.1.9. NetworkManager

Der NetworkManager kümmert sich um die Verwaltung des WLANs. Dabei ist es hier möglich WLAN spezifische Einstellungen wie zb. die Änderung der SSID zu tätigen. Dieses Modul verwaltet nicht den Server selbst, da dies die Aufgabe des Web Server Moduls innerhalb des WebInterface Moduls innerhalb der Communication Ebene ist.

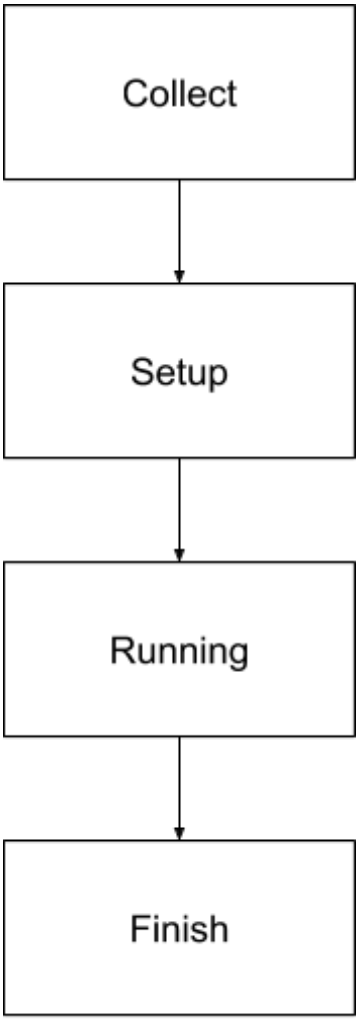
4.2.2. Game-Ebene

Die Game Ebene ist so ziemlich der wichtigste Bestandteil des Systems, da hier das Spiel selbst simuliert wird. während die anderen beiden Ebenen nicht unbedingt abhängig vom Spiel selbst sind, heißt man könnte sie auch in anderen Projekten verwenden und müsste nur die Game Ebene austauschen, ist die Game Ebene sozusagen das Spiel selbst. Es ist wichtig diese Ebene als ein Art Blackbox zu betrachten. Das heißt es wird nur mittels des freigegebenen Interfaces darauf zugegriffen und stellt sich von selbst mithilfe der Device Ebene da. Dabei sind die beiden primären Module der GameController, welcher den ganzen Spielablauf verwaltet und der BoardManager, welcher die jeweiligen Spieldaten mithilfe der Device-Ebene darstellt. Um die Aufgabe und Funktionsweise aller Module in dieser Ebene

verstehen zu können, müssen wir uns zunächst den Spielablauf anschauen. Auf die Module wird nach dem Spielablauf näher eingegangen.

4.2.2.1. Spielablauf

Der Spielablauf ist in vier lineare Teile oder auch Phasen unterteilt, welche hier rechts abgebildet sind. Dabei ist zu beachten, dass hier weder auf die vorherige Initialisierung, noch auf mögliche Fehler eingegangen wird, da diese in der Verantwortung der Device-Ebene liegen.

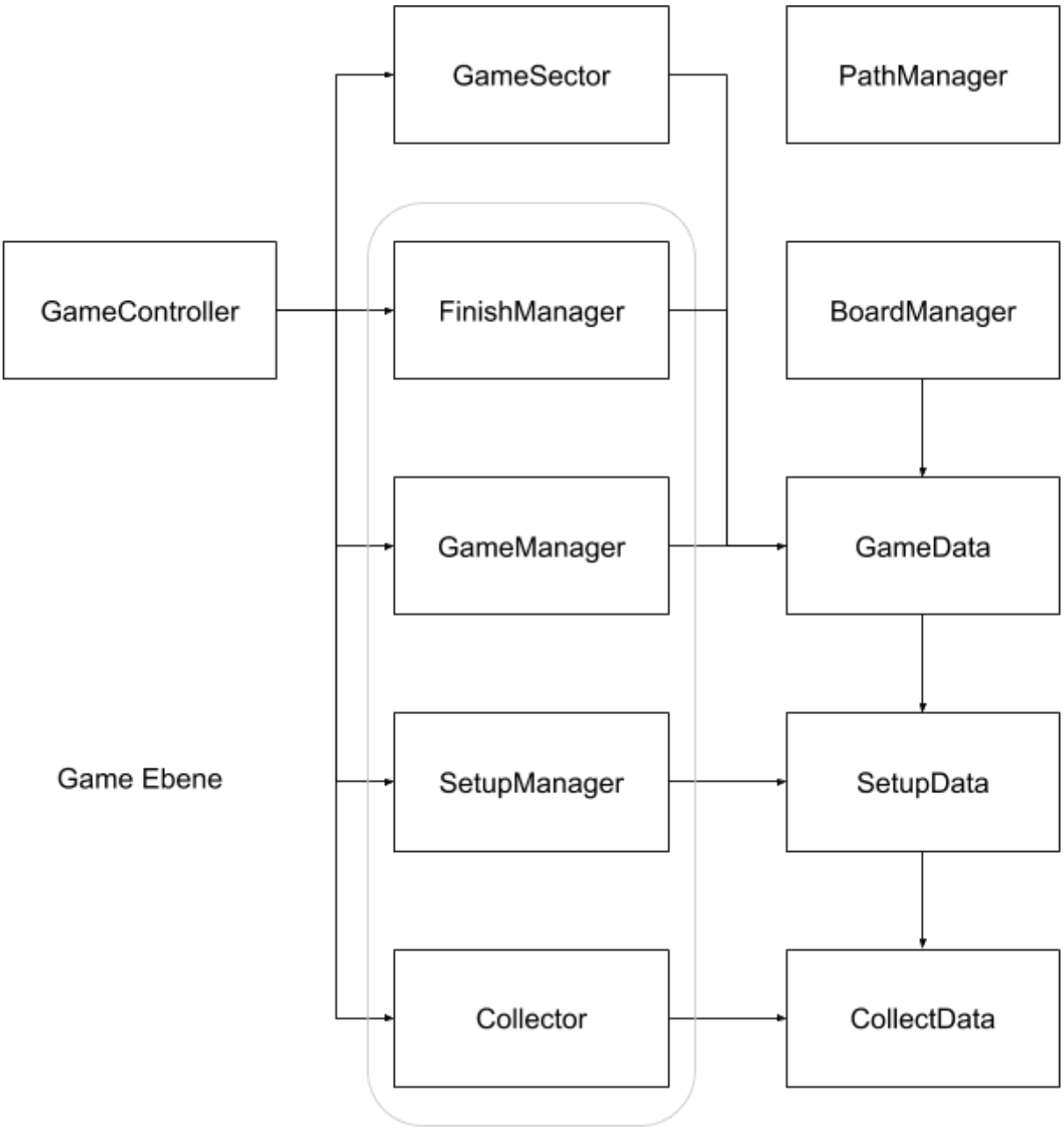


Spielablauf

Ein normales Spiel fängt in der Phase Collect an, in welcher es darum geht, auf alle Spieler zu warten und diese zu “sammeln”. Dabei geschieht in dieser Phase nichts anderes außer der Verwaltung der Spieler. In der Setup-Phase geht es darum, subjektive Einstellungen und Änderungen am Spiel zu treffen wie zb. die Änderung der Anzahl der Tickets oder eine Präferenz der Farbe und Rolle. Dabei ist ab dieser Phase die Anzahl der Spieler Fest und kann auch nicht mehr geändert werden. Um hier noch Anpassungen daran zu treffen, muss das Spiel neugestartet werden. Die Running-Phase ist die zentrale Phase, in welcher das Spielgeschehen selbst passiert. Dabei werden immer nacheinander, den Spielern die Möglichkeit geboten, einen Zug auszuführen. Wenn ein Spieler das Spiel gewonnen hat, folgt die Finish-Phase. Diese Phase dient als praktisches Ende des Spiels, in welchem die Communication-Ebene die Möglichkeit hat, dem Spieler anzuzeigen wer gewonnen hat oder ihm eventuell Daten und Statistiken bereitzustellen. Dabei ist es nicht mehr möglich eine weitere Ebene zu erreichen und das Gerät muss neu gestartet werden, um ein neues Spiel zu beginnen.

Es ist auch möglich einen anderen Ablauf zu nehmen, dabei werden die ersten beiden Phasen übersprungen und gleich bei der Running-Phase angefangen. Dies ist der Fall, wenn ein vorherig beendetes Spiel, wegen eines Fehlers oder einer Pause, wiederhergestellt werden soll. Dafür werden alle Spieldaten automatisch in einem Sektor (siehe 4.2.1.5. MemoryManager) gespeichert und bei jener Wiederherstellung aus diesem geladen. Daraus ergibt sich eine sehr hohe Resistenz entgegen eines Abbruchs eines Spieles.

Alle vier Phase haben ihr eigenes Modul und eigene Datenstruktur. Dabei haben die immer späteren Phasen jeweils Zugriff auf die Daten der früheren Phasen. Der Ablauf der Phasen wird mit Hilfe des GameControllers gesteuert und kann auch manuell weitergeschaltet werden.



4.2.2.2. GameController

Der GameController stellt das zentrale Modul der Game-Ebene da und verwaltet den Ablauf, das Speichern, sowie die Wiederherstellung eines laufenden Spieles. Dabei ist es anderen Modulen möglich die Beendigung der zur Zeit laufenden Phase zu beantragen, welche aber nicht direkt bearbeitet wird, da es sonst zu Fehlern in der Reihenfolge kommen könnte. Bei der Bearbeitung der Anfrage wird das zurzeit laufende Modul zum Beenden gebeten, was es jedoch auch ablehnen kann. Da es für andere Module keinen Weg gibt zu erfahren, ob ihre Anfrage umgesetzt wurde, dürfen diese sich nicht auf einen Erfolg verlassen. Die Wiederherstellung eines Spiels geschieht nur, wenn der vorherige Phase Running war. Dann wird der Nutzer gefragt, ob er das Spiel wiederherstellen wollen würde.

4.2.2.3. GameSector

Der GameSector stellt die konkreten Daten dar, welche in einem Sektor (siehe 4.2.1.5. MemoryManager) gespeichert werden. Dabei enthält er die jetzige Phase des Spiels und alle drei Daten-Konstrukte der drei ersten Phasen.

4.2.2.4. BoardManager

Die zentralen für alle Spieler wichtigen Informationen zu präsentieren ist die Aufgabe des BoardManagers. Dieser wird bei Änderung der Daten oder wenn nötig pro Zeit für Animationen aufgerufen. Dabei stellt er die Informationen durch die Device-Ebene mithilfe der LEDs und des LCDs da. Es ist möglich auf den BoardManager eine LED-Animation zu laden, welche aus Daten der Zeiten, den LEDs und der Farben besteht. Da der BoardManager es nicht bemerkt, ob das LCD von anderen Modulen geändert wurde, muss er benachrichtigt werden, wenn es zu wiederherstellen ist.

4.2.2.5. Collector

Der Collector ist die erste Phase des Spiels und wird immer zu Beginn geladen. Seine einzige Aufgabe ist die Verwaltung der Spieler. Dabei können während seiner Phase Spieler hinzugefügt oder entfernt werden, und ihnen werden automatisch Spieler IDs (zufälliges Byte) zugeordnet. Nach der Beendigung dieser Phase ist es nicht mehr möglich noch weiter Spieler hinzuzufügen oder zu entfernen.

4.2.2.6. CollectData

CollectData beinhaltet die für den Collector wichtigen Daten, welche nur die Spieler-IDs wären und die gesamte Anzahl der Spieler wären.

4.2.2.7. SetupManager

Die zweite Phase des Spiels wird durch den SetupManager dargestellt. Er legt die Reihenfolge der Spieler fest und erlaubt Einstellungen an Daten und Farben dieser. Dadurch kann ein bestimmter Spieler zum bösen gewählt werden oder jemand sich seine Lieblingsfarbe auswählen. Auch noch möglich ist die Anpassung der spielspezifischen Daten wie die Anzahl der Tickets.

4.2.2.8. SetupData

Für den SetupManager wichtige Daten werden in SetupData gespeichert. Diese sind Einstellungen wie die Anzahl der Tickets zu beginn, die Reihenfolge der Spieler und ihre Farben sowie Rolle und die Spieler-ID des Bösen.

4.2.2.9. GameManager

Der GameManager repräsentiert die dritte und wichtigste Phase des Spiels. Dieses Modul erlaubt das Tätigen von Spielzügen, sofern dieser Spieler an der Reihe ist und die Abfrage von Spielerdaten. Dabei geben Anfragen zu Spielzügen bei fehlerhafter Anfrage einen Fehlercode zurück (fehlerhaft heißt zb. nicht möglich oder nicht am Zug), welcher dieses Modul in eine Fehlernachricht umwandeln kann, welche wiederum dem Spieler angezeigt

werden kann. Dabei ist zu beachten, dass Spielzug-Anfragen mithilfe eines Turns getätigt werden. Dieser Turn beinhaltet die Daten, welcher Spieler diesen Zug tätigen will, wohin der Spieler möchte, mit welchem Ticket er dies erreichen will und ob er einen doppelten Zug tätigen will.

4.2.2.10. GameData

In GameData werden während des Spiels relevante Daten für den GameManager gespeichert, welche aus den Spielern, dem Spielstand und dem Ergebnis bestehen. Die Spieler enthalten Daten wie die Anzahl an Tickets, den zurückgelegten Pfad oder die jetzige Position. Außerdem sind dort auch rollenspezifische Daten wie die Anzahl der schwarzen Tickets enthalten. Der Spielstand besteht aus Daten wie der jetzigen Rundenzahl, dem Spieler, welcher am Zug ist und Daten über den Bösen wie seine letzte aufgezeigte Position oder verwendeten Tickets. Im Ergebnis ist bisher nur die gewinnende Rolle angegeben.

4.2.2.11. PathManager

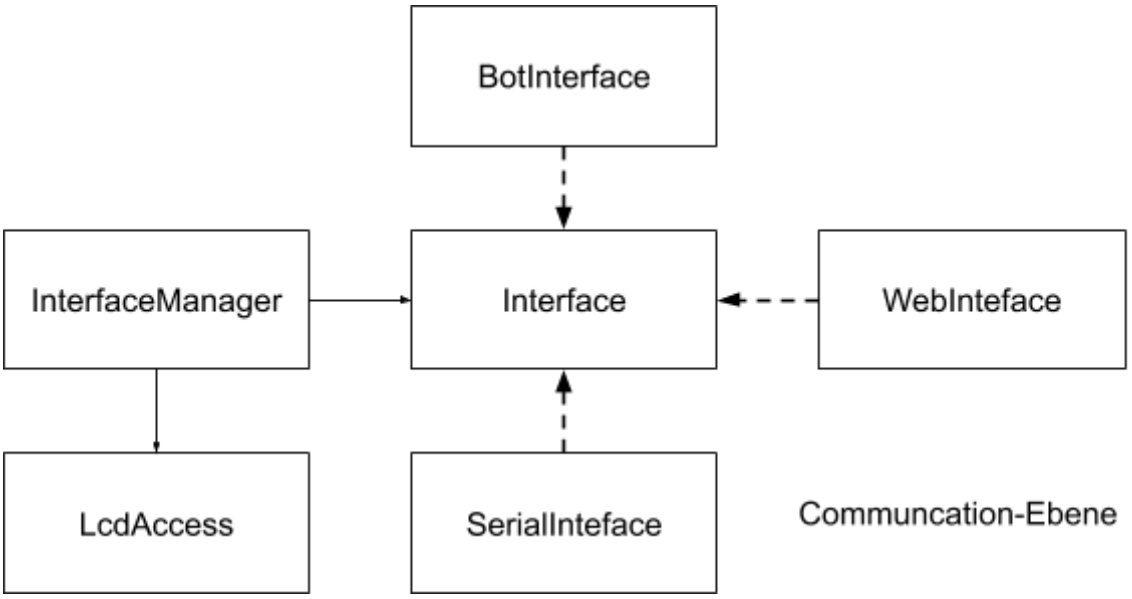
Das Modul PathManager enthält alle Daten der Stationen und deren Verbindungen. Es bietet anderen Modulen die einfache Möglichkeit nach bestimmten Verbindungen von einer Station aus zu Suchen und ermöglicht die leichtere Abfrage von Infos über die Stationen selbst, wie zb. deren Typ. Eine weitere Aufgabe des PathManagers ist es die Startplätze für Spieler zufällig aus einer Liste auszusuchen.

4.2.2.12. FinishManager

Das Modul verwaltet wie der Name es sagt die letzte Phase des Spiels, nämlich das Ende. Dieses Modul gibt bisher keine Funktionen Preis und hat eher den Platz eines Stellvertreters. Die relevanten Dinge dieser Phase geschehen außerhalb, da dort alle Daten schon festgesetzt sind und es nichts gibt, dass der Nutzer am Spiel noch ändern könnte. Der Neustart eines Spiels geschieht ebenfalls außerhalb des FinishManagers und mögliche Animationen und Grafiken gehören ebenfalls nicht zu seinen Aufgaben. Es gab die Überlegung dieses Modul zu entfernen, jedoch ist dies nach Konventionen schwierig, da es immer ein Modul für die jeweilige Phase geben muss. Deswegen kann es jedoch ganz gut sein, dass dieses Modul nicht mehr existieren wird.

4.2.3. Communication

Der einzige Zweck der Communication-Ebene ist die Kommunikation mit dem Nutzer. Dabei wird keine feste Schnittstelle der Kommunikation angenommen, wodurch sich diese Ebene durch beliebig viele erweitern lässt. Der InterfaceManager steuert die Verarbeitung der Schnittstellen (Interface). Die Kommunikation selbst und das Erstellen neue Schnittstellen ist die Aufgabe der Module der Schnittstellen. Zusätzlich gibt es die besondere Art der Kommunikation durch das LCD und drei Knöpfen, welche von LcdAccess gesteuert wird. Nachfolgend ist das Modell dieser Ebene zu sehen.



4.2.3.1. LcdAccess

Alle Eingaben über das LCD und die darunter liegenden Knöpfe werden durch LcdAccess verwaltet. Dabei sind die Menüs, welche LcdAccess zur Verfügung stellt abhängig von der zurzeit laufenden Phase. Die Menüs funktionieren ähnlich wie ein Stapel, wenn man im dritten Untermenü ist, muss man auch dreimal das Menü schließen, um wieder rauszukommen. Alle Menüs und Untermenüs sind leicht erweiter- und anpassbar. Wichtig noch: Währenddessen das Spiel ein Menü geöffnet hat, kann kein Zug weiterverarbeitet werden, da LcdAccess nicht asynchron programmiert wurde. Währenddessen Code in LcdAccess ausgeführt wird ist somit die meiste Verarbeitung blockiert. Ausgenommen ist die Verarbeitung von Web-Anfragen durch den Web- oder Websocket Server, da diese auf einer eigenen asynchronen Bibliothek basieren.

4.2.3.2. InterfaceManager

Das Modul InterfaceManager verwaltet die Erstellung und die Ausführung aller Interfaces. Dabei werden alle Interfaces der Reihe nach verarbeitet und wenn sich Daten in der Game-Ebene geändert haben, werden alle Interfaces “erneuert”. Es können theoretisch Interfaces jeglicher Art existieren, da der genaue Typ des jeweiligen Interfaces nicht bekannt ist (siehe 4.2.3.3. Interface). Dazu gibt zwei Arten der Erstellung von Interfaces, diese wären “pushed” oder “linked”. Der einzige Unterschied liegt darin, dass das erste einen neuen Spieler mit erstellt und das letzte nur ein neues Interface an den als Parameter übergebenen Spieler bindet. Die Bindung an einen bereits bestehenden Spieler ist in dem Sinne wichtig, da nach der Collect-Phase keine neuen Spieler mehr hinzugefügt werden können. Es kann sein, dass es in Zukunft nicht mehr möglich sein wird, ein Interface ohne gebundenen Spieler außerhalb der Collect-Phase zu erstellen. Bisher führt dies jedoch zu undefiniertem Verhalten.

4.2.3.3. Interface

Das Modul Interface stellt eine Art Schnittstelle von Interfaces dar und kann eher wortwörtlich als Interface, als ein Modul gesehen werden. Das Interface ist die einzige Ausnahme im ganzen Projekt, bei welcher Klassen genutzt werden. Es stellt mehrere virtuelle Funktionen zur Initialisierung und Verarbeitung bereit, dazu ermöglicht es auch noch die Weitergabe von Fehlern (siehe 4.2.1.7. FaultHandler). Das Interface wird durch Polymorphismus im InterfaceManager genutzt, wodurch es keine Begrenzung dafür gibt, welche Interfaces existieren müssen. Ein konkretes Interface muss alle Funktionen überschreiben und sich falls nötig im InterfaceManager registrieren.

4.2.3.4. BotInterface

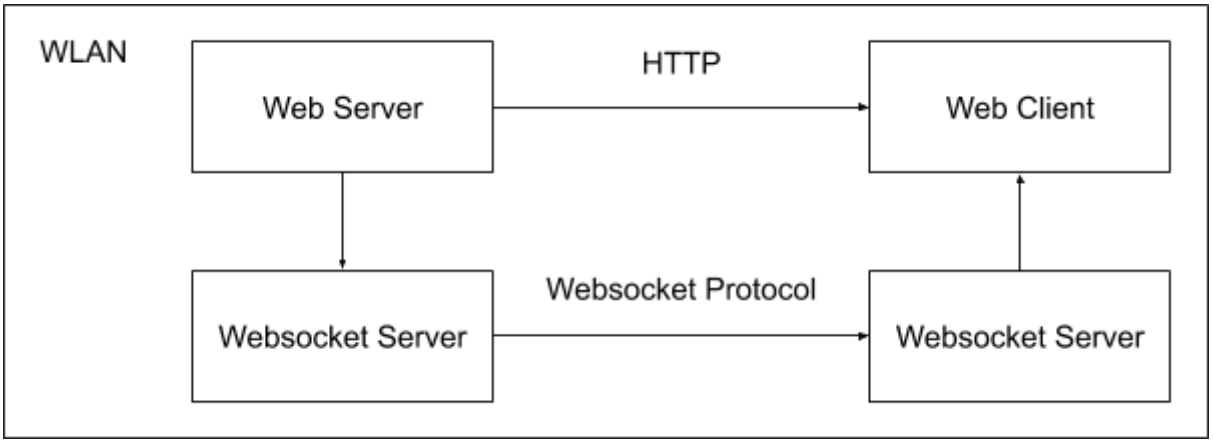
Zur Zeit der Fertigstellung dieses Dokuments ist dieses Modul noch nicht implementiert. Weitere Details sind unter 5.3.1. Künstliche Intelligenz zu finden.

4.2.3.5. SerialInterface

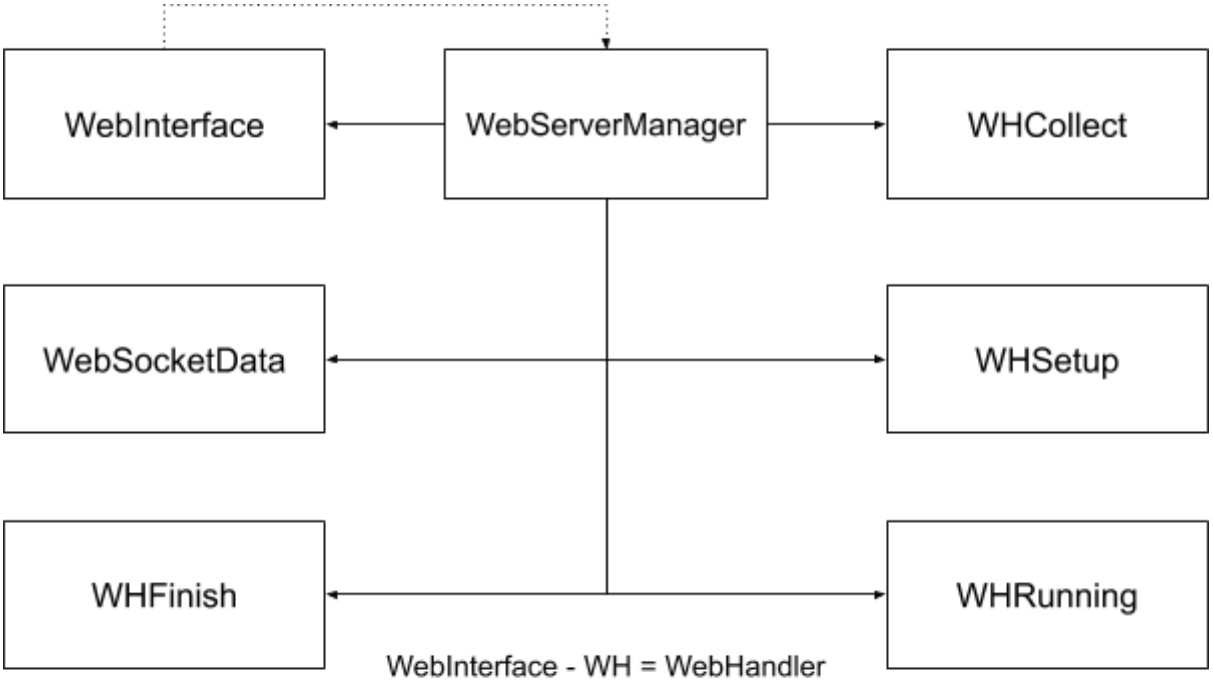
Das SerialInterface dient zur Kommunikation mit dem Spieler über eine Serielle Schnittstelle. Es dient zu Beginn nur zu Testzwecken und wird auch nicht weiterentwickelt. Es kann sein, dass es zur Zeit der fertigstellung dieses Dokuments nicht mehr funktionsfähig sein wird. Stattdessen soll das WebInterface oder das in Zukunft geplante TcpInterface genutzt werden.

4.2.3.6. WebInterface

Das WebInterface ist die zentrale Zugriffsmöglichkeit unseres Projekts. Sie bietet zu allen Phasen ein leicht zu bedienendes Interface, welches als Website im WLAN bereitgestellt wird. Da das WebInterface an sich ein größeres Thema ist, werden wir hier ein wenig näher auf die Funktionsweise und alle einzelnen Komponenten eingehen. Das Modul WebInterface selbst dient als Schnittstelle zwischen dem WebServer, dem Websocket Server und dem InterfaceManager oder auch dem Spiel.
In der folgenden Abbildung ist das Konzept des WebInterfaces zu sehen. Es wird ein Webserver und ein Websocket Server auf dem Port 80 gestartet. Es ist beides auf dem gleichen Port möglich, da unsere Web-Bibliothek asynchron arbeitet. Der Webserver verarbeitet dabei die Anfragen des Web Clients und sendet ihm eine Website. Die Identifizierung von Nutzern handhaben wir durch Cookies, welche sich ähnlich wie Variablen



nutzen lassen. Wenn bei der Anfrage der Website entweder kein oder ein ungültiges Cookie vorhanden ist, wird ein neuer Nutzer, falls wir uns in der Collect-Phase befinden, erstellt und dem Nutzer ein gültiger Cookie gesendet. In allen anderen Phasen wird man zu einer “RequestPID” Website weitergeleitet, dazu später mehr. Falls jedoch ein gültiger Cookie vorhanden ist, wird dieser zu dem Spieler gebunden. Diese nun gesendete Website enthält ein Script, welches eine Websocket-Verbindung zum Websocket-Server des Boards aufbaut. Diese Verbindung wird nun vom Board empfangen und zum WebInterface gebunden. Falls nach erfolgreichem senden einer Website nach kurzer Zeit keine Verbindung zum WebInterface mittels Websocket erstellt wird, wird dieses Interface verworfen und gelöscht. Nach finaler und erfolgreicher Ausführung all dieser Schritte sendet das WebInterface ab sofort bei Änderung der Daten eine Nachricht mittels Websocket an den Nutzer, welcher die Website aktualisiert.
Nun wird nochmal kurz auf alle Module, die hinter dem WebInterface liegen, eingegangen. In der folgenden Abbildung sind die Abhängigkeiten dieser zu sehen.



4.2.3.7. WebServerManager

Dieses Modul ist die primäre Komponente der Darstellung der Website. Es verwaltet den Webserver sowie den Websocket-Server und alle nötigen Module zur Verarbeitung und Darstellung der Website. Ein WebInterface meldet sich bei der Erstellung an oder bei der Entfernung ab. Da alle Webinterfaces hier angemeldet sind, können andere Module hier nach Interfaces durch eine Player-ID suchen.

4.2.3.8. WebHandler

Es gibt kein einzelnes WebHandler-Modul, sondern eine Reihe an Modulen, welche alle zur Gruppe der WebHandler gehören. Da diese sich nicht sehr stark unterscheiden, werden sie hier zusammengefasst. Ein WebHandler behandelt die Anfrage eines Nutzers, wobei ein

einzelner WebHandler immer eine einzelne Adresse behandelt. Hierbei werden die Anfragen entweder beantwortet, umgeleitet oder abgelehnt. Zu beachten ist, dass auch abgelehnte Anfragen, den Nutzer nicht aus dem WLAN rauswerfen. Antworten werden mit Hilfe des WebConstructors konstruiert.

4.2.3.9. WebConstructor

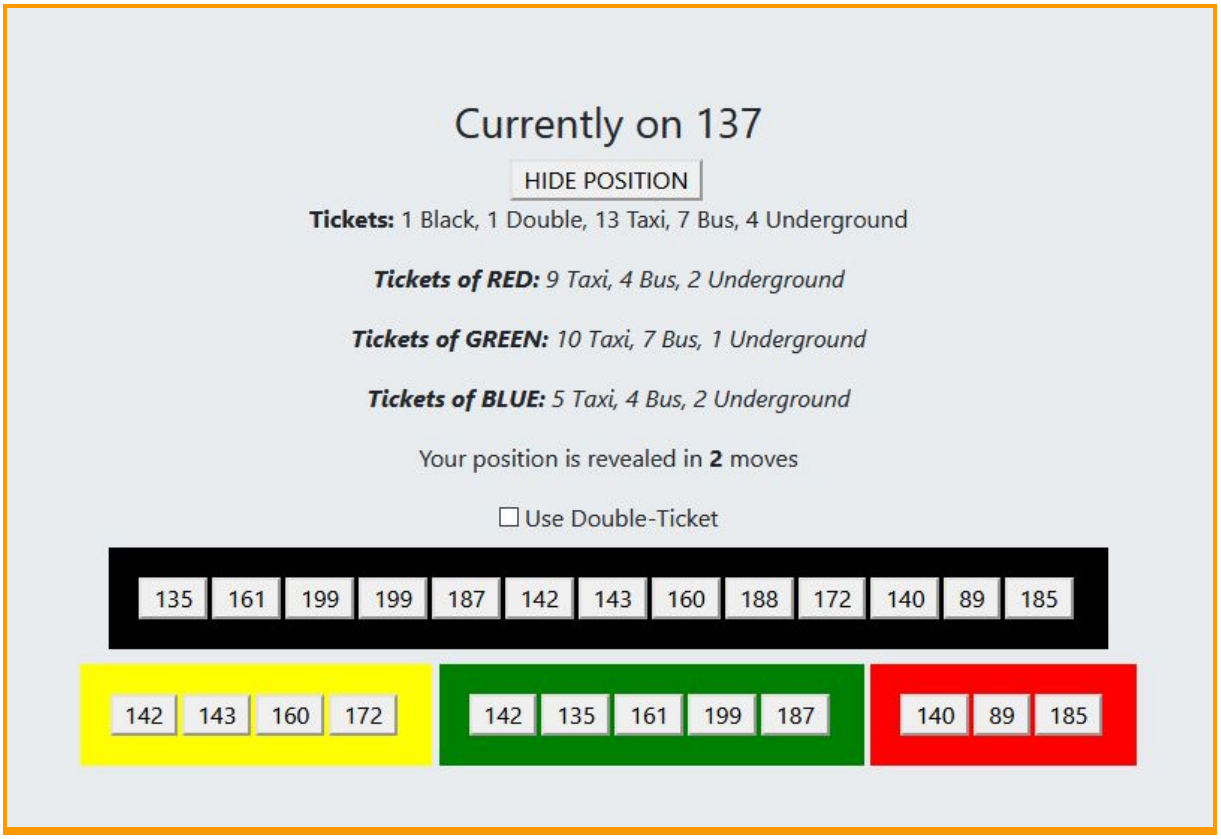
Der WebConstructor wurde nicht in das oben gezeigte Modell eingebunden, da er keine wichtige Rolle in der Abhängigkeit darstellt. Alle WebHandler nutzen dieses Modul zur Herstellung der Website, dabei wird eine Website aus einer Liste von Teilen hergestellt. Es ist jedoch auch möglich eigene dynamische Teile einzubinden, da die normalen Teile alle vorher festgelegt sind und nicht geändert werden können. Durch das Einbinden eigener Teile entsteht ein größerer Aufwand, welcher vermieden werden sollte.

4.2.3.10. WebSocketData

Dieses Modul funktioniert ähnlich wie die Datenmodule der Game-Ebene. Es enthält alle Daten zur Nutzung der Kommunikation zwischen dem Websocket und dient sozusagen als Protokoll dieser Kommunikation.

4.2.4. Website

Die Websites dienen als primäre Schnittstelle der Kommunikation zwischen dem Spieler und dem Spiel, weswegen sie besonders benutzerfreundlich und intuitiv gestaltet wurden. Die meiste Zeit werden alle Vorgänge bereits vom Backend behandelt, bzw. vom Frontend unterstützt. Dabei haben wir unsere Website nicht als Ordnerstruktur, entwickelt sondern im Gegensatz als Monolith, da die Bereitstellung weiterer Pfade unnötige Performanceverluste mit sich ziehen würde und wir einen anderen modularen Weg zur Websitegenerierung gefunden haben (siehe 4.2.3.9. WebConstructor). Die Website besteht meistens aus einem Gerüst, welches sein Benutzerinterface automatisch mit JavaScript und dem Websocket füllt.



4.2.4.1. Skript

Zur Darstellung der durch den Socket kommunizierten Daten, generiert das Skript durch einen Algorithmus HTML-Code. Dieser kategorisiert die Züge basierenden auf der Wegart und entfernt alle Züge, die nicht durchgeführt werden könnten. Die Daten, werden dabei zusätzlich in die Anzeige der verfügbaren Tickets verarbeitet. Zudem bietet es dem Villian/Gegner, die Möglichkeit seine aktuelle auf dem Bildschirm angezeigte Position vor neugierigen Blicken zu schützen, also wenn z.B. gerade ein Detektiv hinter einem steht, indem diese mittels eines Buttons temporär ausgeblendet werden kann. Betätigt ein Spieler einen Button wird dieser Zug unter Beachtung der möglichen Sonderattribute (z.B. der Doppelzug), mittels einem POST-Request übermittelt (näheres in 4.2.5.2.1. Zuganfrage durch Website).

4.2.4.2. Modularität

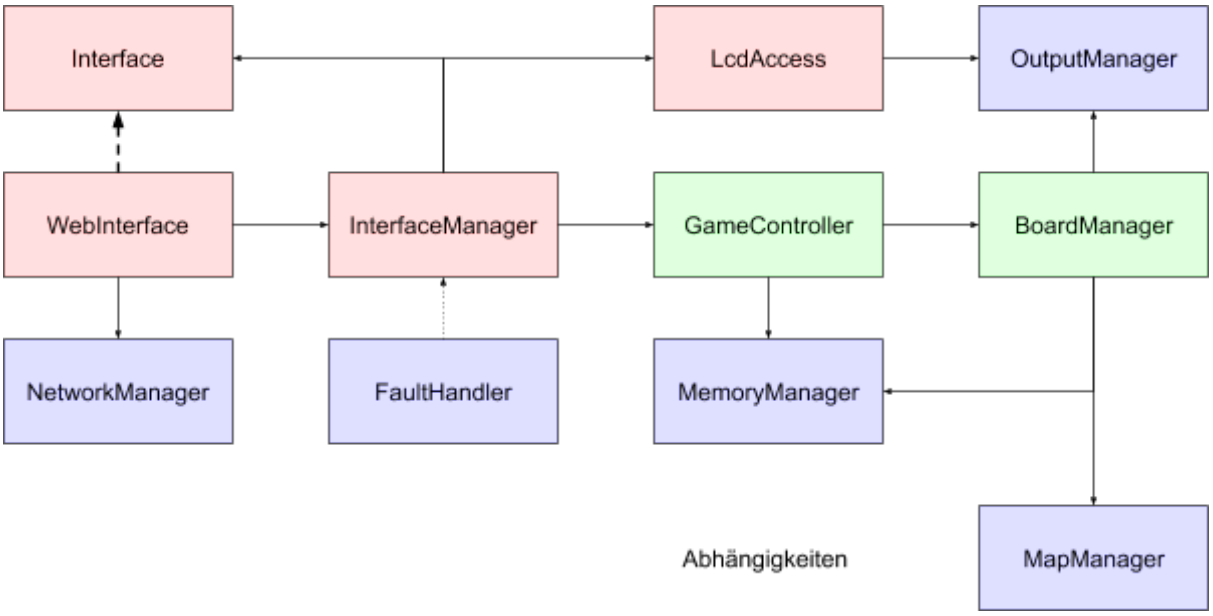
Die einzelnen HTML-Dateien sind modular aufgebaut, sodass aufgrund der Zerteilung redundanter Daten, Speicher gespart wird. Dabei sind die einzelnen Module in die üblichen HTML-Strukturen (head, body, script, etc.) aufgeteilt und wurden mit Tools komprimiert.

4.2.5. Abhängigkeiten

Nachdem alle Ebenen und Module bekannt sind behandeln wir in diesem Abschnitt wichtige Abhängigkeiten und Abläufe. Dabei beziehen wir uns weniger auf die Abhängigkeiten und Interaktion der Ebenen, sondern eher auf die, der Module miteinander.

4.2.5.1. Abhängigkeiten

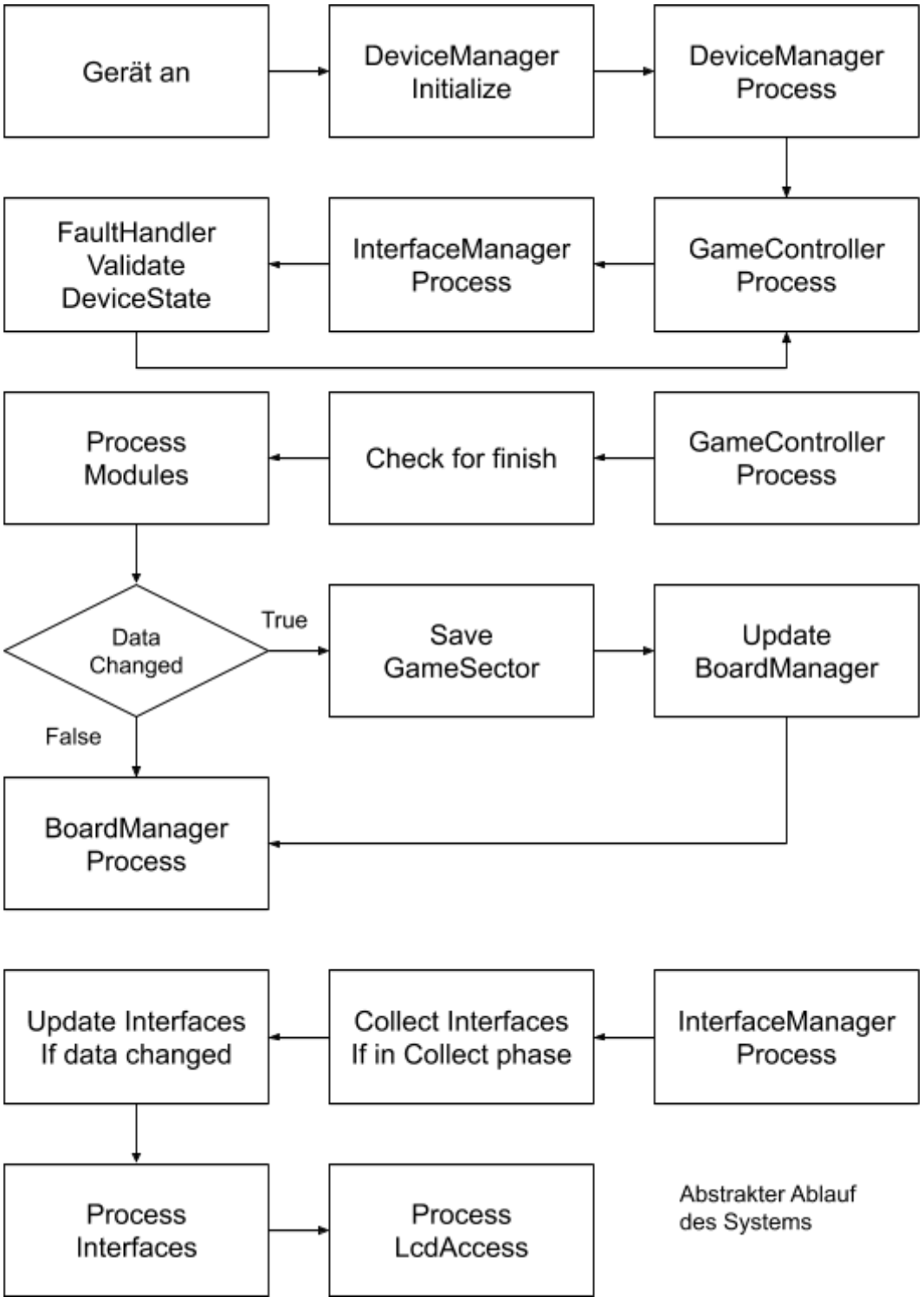
Die wichtigen Abhängigkeiten sind in Folgender Abbildung dargestellt. Dabei ist hier die Communication-Ebene Rot dargestellt, die Game-Ebene Grün und die Device-Ebene Blau. Es ist direkt zu erkennen, dass immer nur die oberen Ebenen auf die niedrigeren Ebenen Zugriff haben, bis auf den FaultHandler. Dort setzt wie schon in vorherigen Abschnitten erwähnt der InterfaceManager ein Callback im FaultHandler, welches der FaultHandler bei Fehlern aufrufen kann. Das WebInterface erbt von Interface und hat Zugriff auf den NetworkManager. Weitere Interfaces wurden nicht dargestellt, da diese in diesem Zusammenhang keine Wichtigkeit haben. Der InterfaceManager verarbeitet Neuerungen in den Daten und erneuert dadurch alle Interfaces. Der GameController prüft intern auf veränderte Daten und speichert bei einer Änderung den neuen Sektor sowie erneuert die Darstellung durch den BoardManager. Der BoardManager konvertiert Positionen des Spielfeldes mit Hilfe des MapManagers und gibt diese dann durch den OutputManager aus. Der InterfaceManager ruft wiederholt LcdAccess auf, welcher auf Eingaben durch das LCD wartet. Falls eine Eingabe angefangen wird, öffnet LcdAccess ein Menü und währenddessen dieses Menü offen ist, sind alle anderen Verarbeitungen unterbrochen (siehe 4.2.3.1. LcdAccess). LcdAccess wiederum stellt ähnlich wie der BoardManager sich selbst mittels OutputManager da.



4.2.5.2. Ablauf

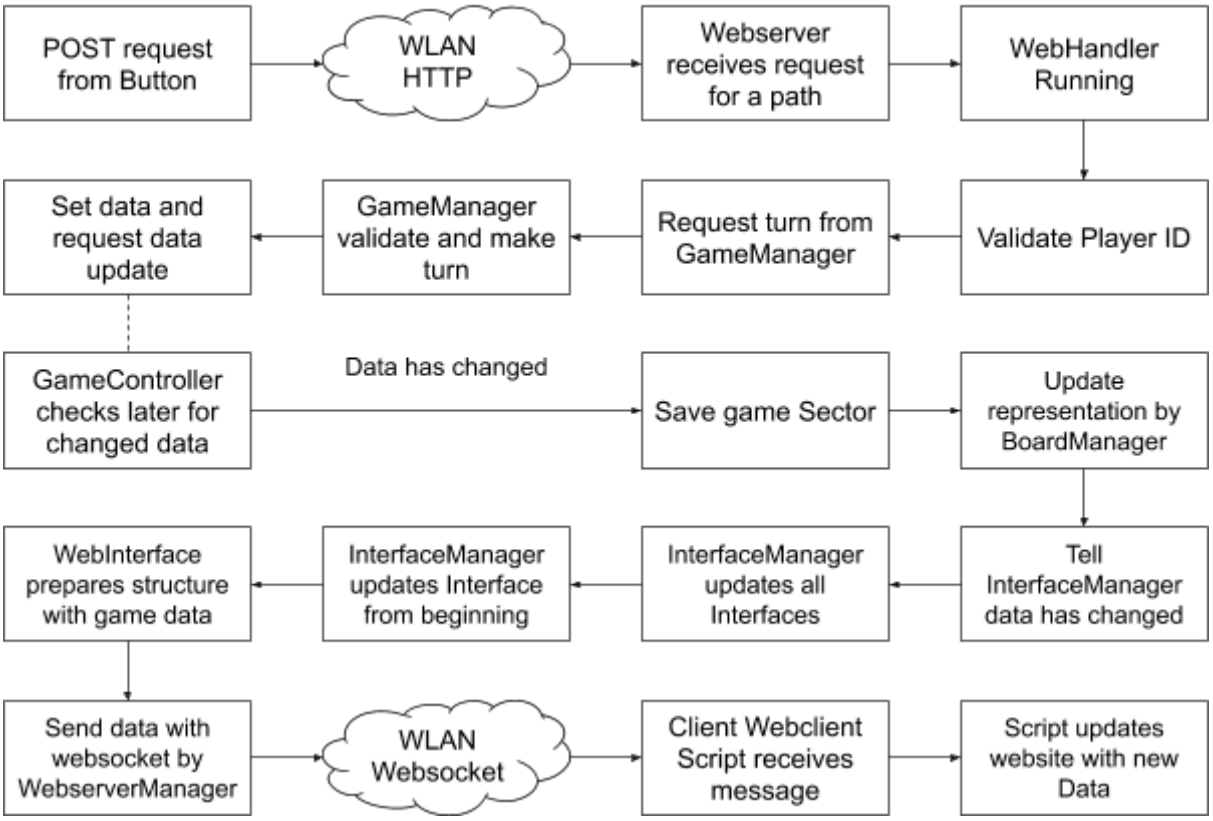
Hier werden nun einige typische Abläufe demonstriert, welche das Verständnis der Architektur stärken sollen. Auf Modelle wurde hier verzichtet, da sich Sequenzdiagramme für einzelne Schritte nicht eigneten und leider keine Erfahrung mit Aktivitätsdiagrammen besteht. Jedoch wird auf der im folgenden Bild mithilfe eines Modells der ungefähre Ablauf des ganzen Systems dargestellt. Dabei wird nicht auf Einzelheiten eingegangen, sondern nur kurz auf die Modulnamen. Auf dieses Modul wird nicht näher eingegangen, da sich die Module in ihren eigenen Beschreibungen, welche vorher alle beschrieben wurden, nun

selbst erklären. Dies dient nur als Zusammenfassung und leichteren Einstieg in die Weiterentwicklung des Systems.



4.2.5.2.1. Zuganfrage durch Website

Ein Beispiel eines Zuges lässt sich in dieser Weise beschreiben. In der folgenden ist zusätzlich dieser Prozess illustriert. Zu Beginn warten die Interfaces, der InterfaceManager und der GameManager auf eine Aktion eines Nutzers. Wenn nun ein Nutzer eine Anfrage für



einen Zug sendet, heißt er drückt einen Knopf, wird eine POST Anfrage von dem Webclient an den WebServer gesendet. Ein WebHandler verarbeitet nun diese Abfrage und validiert die PlayerID. Falls die Player ID nicht richtig sein sollte, wird der Nutzer zur Website RequestPID verwiesen. Mithilfe dieser PlayerID und der ihm gesendeten Daten durch die POST Methode gibt er eine Anfrage für einen Zug im GameManager auf. Diese Anfrage wird nun sofort vom GameManager beantwortet und bei einem Fehler direkt dem Spieler mit einer Fehlernachricht des GameManagers durch den Fehlercode durch eine Antwort dargestellt. Bei erfolgreicher Anfrage wurde der Zug getätigt und die Daten in GameData aktualisiert. Da dadurch auch Daten von GameSector geändert wurden, wurde eine Anmerkung im GameManager gesetzt. Später verarbeitet der GameController den GameManager und bekommt mitgeteilt, dass sich die Daten geändert haben. Da sich die Daten nun geändert haben wird der GameSector als Sektor gespeichert (siehe 4.2.1.5. MemoryManager) und die Darstellung des Spiels (LEDs) durch den BoardManager aktualisiert. Danach wird wiederum dem InterfaceManager zurückgegeben, dass sich die Spieldaten geändert haben und alle Darstellungen aller Interfaces erneuert werden müssen. Das wiederum wirkt bei dem WebInterface von unserem Client von Beginn an aus, dass eine neue Struktur nach der WebSocketData erstellt und ihm gesendet wird. Diese kommt bei ihm an, wird verarbeitet und in seiner Website dargestellt.

4.3. Konventionen

Um strukturierten und lesbaren Code zu erreichen folgen wir soweit wie möglich festen Konventionen. Dabei werden wir uns auf das Google Style Guide für C++ beziehen. Weitere Besonderheiten und Anpassungen, welche wir speziell für unser Projekt benötigen, werden hier noch besprochen.

4.3.1. Dateistruktur

Ordner sind zur Gruppierung von Modulen da und sollten nicht außerhalb der festen Ebenen erstellt werden.
Alle Einbindungen von Header gehören, wenn möglich in Quelldateien um die Kompilierzeit zu kürzen und versehentliche gegenseitige Abhängigkeiten zu vermeiden. Zirkuläre Abhängigkeiten sollten allgemein vermieden werden, um die Skalierbarkeit zu erhalten und die Komplexität zu senken.

4.3.2. Modulstruktur

Auf Klassen soll, wenn möglich verzichtet werden. Stattdessen sollen Module durch gruppierte Funktionen in Namensräume definiert werden. Dabei sollte ein Modul nur solche Funktionen freigeben, welche für andere Module wichtig sind oder die erhebliche performanceorientierte Vorteile bieten. Wichtig ist dass, jedes Modul innerhalb einer Ebene existieren muss woraus folgt, dass dieses Modul auch innerhalb des Namensraumes der jeweiligen Ebene geschrieben werden muss.

4.3.3. Makros

An Makros sollte in der Regel erkennbar sein, zu welcher Ebene sowie Modul sie gehören. Außerdem soll anstatt wie im Google Style Guide vorgeschlagenen ‘include guards’, das Microsoft spezifische ‘pragma once’ genutzt werden.

4.3.4. Speicherplatz

Generell soll auch auf dynamische Speicherbeschaffung in jedem Fall verzichtet werden, um Speicherfragmentierung zu vermeiden. Falls benötigt, kann welcher vom Speichermanagement zu Verfügung gestellt werden. Daraus folgt auch, dass keine Bibliotheken genutzt werden sollen, wenn diese nicht absolut erforderlich sind. Bibliotheken wie String sind unter allen Umständen zu Vermeiden und dürfen nicht genutzt werden.

4.3.5. Definition von Text

Generell sollte kein Text außerhalb des Flash-Speichers existieren. Dafür werden jeweilige Makros vom Arduino oder speziellen Modulen bereitgestellt. Text für das LCD oder den FaultHandler soll immer mit dem jeweiligen im Modul definierten Makro erstellt werden.

4.3.6. Bibliotheken

Da auf globale Variablen außerhalb der Main, aus uns nicht erkennbarem Grund nicht zugegriffen werden kann, soll ein Zeiger auf jeweilige Objekte in der Main im Namensraum “Extern” definiert sein, auf welchen dann nötige Module, extern über den Namen darauf zugreifen können. Weitere Aspekte zu Bibliotheken sind unter Speicherplatz aufgelistet.

4.4. Fehler

Das System bietet eine zweistufige Fehlerbehandlung, welche der FaultManager und der FailureManager wären. Dabei dient der FaultManager für Fehler, welche oberhalb der Device-Ebene, bzw. oberhalb des FaultManagers geschehen. Deswegen ist die Aufgabe des FailureManagers, Fehler der Module FaultManager, MemoryManager und OutputManager zu handhaben. Diese Fehler kommen seltener vor und werden mithilfe einer RGB Status-LED angezeigt, wobei anzumerken ist, dass nach dem Anzeigen, das Gerät neustartet. Die Codes werden mithilfe der Farbe und der Anzahl des Blinkens dargestellt. Im Gegensatz dazu stellt der FaultManager Fehler mithilfe des LCD dar. Diese können fatal sein, und dazu führen, dass sich das Gerät wie bei dem FailureHandler neustarten wird. Nicht fatale Fehler nennen wir Warnungen. Es gibt eine weitere Fehlerquelle, welche der PanicHandler des ESP32 wäre. Dieser kümmert sich um Gerätespezifische Fehler, welche wir weder verhindern noch korrekt anzeigen können. Deswegen kann es bisher sein, dass das Gerät ohne erkenntlichen Grund abstürzt und neustartet. Da es uns bisher nicht möglich ist den Fehler dann Ausgeben zu lassen, geschweige denn überhaupt auf diesen zuzugreifen / auszulesen, muss dies leider hingenommen werden. Die einzige Möglichkeit diesen Fehler auszulesen wäre über die serielle Schnittstelle, über welcher dieser ausgegeben wird.

4.4.1. Faultmanager als Fehlerbehandler

Fehler als blinkende RGB-LED in bestimmten Farben. Die Anzahl des blinken wird als Fehlercode gesehen.

Farbe	Bedeutung
Rot	Fehler des MemoryManagers.
Grün	Fehler des OutputManagers.
Blau	Fehler des FaultManagers.
Lila	Unbekannter Fehler. Typischerweise weist dies auf einen Fehler in der Fehlerbehandlung selbst hin.
Gelb	In jedem Fall unbehebbarer Fehler. Software wurde in falscher weise geändert und muss zum vorherigen Zustand gesetzt werden. Gerät startet nicht neu.
Türkis	Fehler in der Fehlerbehandlung. Ein zu hoher Fehlercode wurde erhalten.

4.4.1.1. MemoryManager

Fehlercode	Bedeutung
2	“HEAP_OVERFLOW” weist auf eine zu große dynamische Beschaffung von Speicher. Der MemoryManager war nicht in der Lage einen in der angeforderten länge großen freien Block zu finden.
3	“INVALID_SECTOR”: Ein Modul versuchte auf einen ungültigen Sektor zuzugreifen. Dies ist außerdem ein Fehler der Farbe Gelb.
4	“EEPROM_OVERFLOW” weist auf einen ungenügenden EEPROM zur Laufzeit. Das Makro DEVICE_EEPROM_SIZE ist zu erhöhen.
5	“EEPROM_CORRUTED”: Der EEPROM weist eine ungültige Signierung auf und wird deswegen gelehrt und neu signiert. Dieser Fehler sollte nach dem Neustart nicht mehr auftreten.
6	“EEPROM_WRITE_CLEAN”: Es gelang nicht den EEPROM zu lehren. Dies ist bisher noch nie aufgetreten und sollte niemals auftreten, weswegen wir bisher auch keine Lösung für ein mögliches auftreten haben.
7+	“EEPROM_WRITE” zeigt ähnlich wie der “EEPROM_WRITE_CLEAN” auf einen Fehler beim Schreiben der Daten. Der Fehlercode besteht dabei aus sieben plus die ID des Moduls. Dieser Fehlercode sollte ebenfalls nicht auftreten.

4.4.1.2. OutputManager

Fehlercode	Bedeutung
3	“LCD_DISPLAY” sollte nicht als Fehlercode auftauchen, da dieser noch nicht in richtiger Weise umgesetzt werden konnte.

4.4.1.3. FaultHandler

Fehlercode	Bedeutung
5+	Diese Fehlercodes weisen auf Fehler des FaultHandlers, welche eventuell nicht auf dem LCD nicht angezeigt werden konnten. Dabei steht der Fehlercode für fünf plus das Modul, welches den Fehlercode hatte.

4.4.2. Failuremanager als Fehlerbehandler

Fehlercodes werden primär auf dem LCD angezeigt. Dabei werden sie auf den vier Zeilen und 20 Zeichen in folgender Weise dargestellt. Auf der ersten Zeile sind der Fehlertyp und die Fehlernachricht zu sehen. Dabei spannt sich diese Nachricht auf die zweite Zeile und umfasst maximal 33 Zeichen. Auf der dritten Zeile wird der Modulname dargestellt, welcher maximal 12 Zeichen lang sein darf. Auf der letzten Zeile wird ein bis zu neun Zeichen langer Fehlercode in Hexadezimal angezeigt.
Es gibt Fehler, welche **Fatal** sind. Diese verursachen einen Neustart und sind am LCD unter Error zu erkennen. Bei nicht fatalen Fehlern wird kein Neustart durchgeführt und sind unter Wrnng am LCD zu erkennen.

Modul	Fehlercode	Bedeutung
Interface Manager	2 (Fatal)	“REMOVE_INVALID_PLAYERID”: Es wurde versucht ein nicht existierendes Interface zu entfernen. Das Interface wurde bereits entfernt oder die Spieler ID ist ungültig.
WebHandler Running	2	“INVALID_POST”: Es wurde versucht einen ungültigen Spielzug durch eine POST Anfrage durchzuführen. In der Regel ist dies auf eine Manipulierung der POST-Nachricht durch einen Spieler zurückzuführen, da die POST-Anfragen normalerweise von einem Skript kontrolliert werden. Sonst kommt ein Fehler des Skripts in Frage.
FaultHandler	2 (Fatal)	“INC_NULL”: Das callback zum InterfaceManager wurde nicht gesetzt und ist NULL. Dieser Fehler ist bisher nicht aufgetreten, würde jedoch auf eine fehlende Initialisierung des InterfaceManager hinweisen
FaultHandler	3 (Fatal)	“LOW_MEMORY” weist den Nutzer auf einen Mangel an heap Speicher, weswegen das Gerät sofort neugestartet werden muss.
FaultHandler	4 (Fatal)	“HEAP_FRAG” weist den Nutzer auf einen fragmentierten Speicher, weswegen das Gerät sofort neugestartet werden muss.
Network Manager	2 (Fatal)	“SOFT_AP_MODE” weist auf einen Fehler beim Setzen des Modus des WLANs.
Network Manager	3 (Fatal)	“SOFT_AP_CONFIG” weist auf einen Fehler beim Setzen der Konfiguration des WLANs.

Network Manager	4 (Fatal)	“SOFT_AP_CREATE” weist auf einen Fehler bei der Erstellung des WLANs hin.
Collector	2 (Fatal)	“PLAYER_OVERFLOW”: Es wurde versucht einen Spieler über der maximalen Anzahl an Spielern zu erstellen.
Game Controller	2	“INVALID_FINISH”: Es wurde in einem während einer ungültigen Phase versucht diese zu beenden.
Game Controller	2+ (Fatal)	“INVALID_STATE_RUNNING”: Während des Spieles wurde eine ungültige Phase des Spiels gefunden. Der Fehlercode minus zwei gibt die ID der Phase an.
Game Controller	0xB0+	“INVALID_STATE” weist auf einen invaliden Status im Game Sektor nach einem Neustart hin. Der gefundene Status kann als zweite Zahl des Fehlercodes ausgelesen werden. Der Sektor wird mit neuen Daten eines leeren Spiels überschrieben.

4.5. Bibliotheken

Wir haben für unser Projekt ESP32 spezifische Bibliotheken genutzt, welche bei einem Hardwarewechsel alle ausgetauscht werden müssen. Dabei handelt es sich um Handhabung des Webservers und der WS2812B LEDs.

4.5.1. ESPAsyncwebserver

Zur Bereitstellung eines Web und Websocket Servers nutzen wir die Bibliothek ESPAsyncwebserver, welche für den ESP32 sowie den ESP8266 verfügbar ist. Diese Bibliothek bietet einen asynchronen Server, was ebenfalls ermöglicht, dass der Webserver und der Websocket-Server den gleichen Port nutzen. Jedoch haben wir häufige Probleme bei zur gleichen Zeit ankommenden Anfragen erlebt. Dieses Problem lässt sich jedoch nicht verhindern, da wir sehr stark abhängig von dieser Bibliothek sind. Diese Bibliothek steht standardmäßig nicht in der Arduino IDE zu Verfügung.

4.5.2. FastLED

Für die Verwaltung der WS2182B LEDs nutzen wir die Bibliothek FastLED, welche einen leichten Zugang zu diesen bietet. Sie steht standardmäßig in der Arduino IDE zu Verfügung und unterstützt viele Geräte. Wir haben uns für diese entschieden, da sie kompakter als andere populäre Bibliotheken ist. Dennoch hatten wir häufige Probleme, welche sich durch Makros, bzw. der Exportierung der Initialisierungsfunktion in die Main.ino beheben ließen.

4.5.3. LCD I2C

Für die Ansteuerung eines LCD über einen I2C gab es nur wenige Möglichkeiten, wobei wir uns für die Bibliothek “LiquedCrystalDisplay_I2C” entschieden haben. Diese erlaubt die

Ansteuerung ähnlich wie bei normalen LCD, wobei zusätzlich die Pins für die Serielle Kommunikation verlangt werden.

5. Projektergebnisse

In diesem Abschnitt gehen wir auf die Entwicklung, den jetzigen Standpunkt, den Problemen sowie der zukünftigen Planung ein.

5.1. Funktion

Das jetzige Projekt funktioniert bereits in voller Weise und besitzt ein Gehäuse wie es in L3 beschrieben wurde. Die Technik funktioniert ebenfalls, wie sie im Technologieschema erklärt wurde und muss nur noch in das Gehäuse eingebaut werden. Das Spiel wurde bereits mehrfach gespielt und ist vollkommen und ohne Unterbrechungen spielbar. Ausgenommen sind unbehebbare Abstürze, welche aber durch die Möglichkeit, Spiele erneut laden zu können irrelevant sind. Das Menü kann mithilfe der drei Knöpfe erreicht werden und ist im beschriebenen Umfang nutzbar. Die Website wird dem Nutzer geschickt und funktioniert ebenfalls vollkommen in beschriebener Weise. Die Verbindung und die Aktualisierung durch das Websocket funktioniert dabei auch vollkommen ohne Probleme. Es muss zurzeit noch eine Stromversorgung wie der Laptop genutzt werden, da die Batterien zum Zeitpunkt des Verfassens dieses Dokumentes noch nicht vorhanden sind. Jedoch wurde dies ebenfalls mit einer anderen 5V externen Stromquelle getestet und ist vollkommen funktionsfähig. Alle LEDs des Brettes sind korrekt verkabelt und funktionieren in allen möglichen Farben.
Im Ordner der Bilder, welcher zu diesem Dokument beiliegen sollte, können einige dieser Funktionen begutachtet werden.

5.2. Probleme

Unser Projekt ist nicht ohne Probleme abgelaufen, von welchen wir hier auf einige zentrale eingehen werden.

5.2.1. Löten

Das Verlöten der fast 200 LEDs für das Main-Board, verursachte aufgrund der geringen Erfahrung mit dem Löten im Allgemeinen zu einigen Problemen. Zum einen führte der zu große Durchmesser der Kabel, zu erheblichen Komplikationen beim Löten, da die Kabel mit geschmolzenem Lötzinn dicker waren als die eigentlichen Lötstellen und somit einen Kurzschluss verursacht hätten. Durch die zu lange Erhitzung der Kontaktstellen der LEDs wurde zudem der Heißkleber wieder aufgeschmolzen was wiederum Probleme bereitete. Dank eines von uns bekannten Technikers, der langjährliche Erfahrung mit dem Löten hat, und der uns einen kurzen Lötkurs gegeben hat, konnten wir die Probleme beseitigen. Zur effizienten Verlötung haben wir die einzelnen Lötstücke geplant vorbereitet, um uns auf die einzelnen Arbeitsschritte in diesem Prozess konzentrieren zu können.

5.2.2. WebInterface

Wir standen einmal sehr kurz davor das ganze Webinterface zu streichen und stattdessen alternativen wie externe Programme zu nutzen. Das Problem war, dass der ESP8266 ohne erkenntlichen Grund abstürzte, als wir uns mit mehreren Clients verbunden hatten. Tatsächlich lag dies zum einem daran, dass der ESP8266 maximal 4 Verbindungen gleichzeitig handhaben kann, aber auch an einem weiteren Punkt, auf welchen in ESP8266 eingegangen wird.

5.2.3. ESP8266

Eines der größten Probleme war die Leistung des ESP8266, welchen wir vor dem ESP32 genutzt hatten, welcher nicht genug Speicher hatte, um ein komplettes Spiel und alle Nutzer gleichzeitig handhaben zu können. Deswegen mussten wir auf den ESP32 umsteigen und die ganze Software auf diesen Portieren. Dies war besonders schwer, da der ESP32 über andere Bibliotheken verfügt als der ESP8266. Jedoch kam uns dort die Portierbarkeit der Software zugute (welche zu dem Zeitpunkt bereits **sehr** fortgeschritten war), wodurch es uns nicht zu schwer war, auf den ESP32 zu wechseln.

5.2.4. FastLED

Da Arduino weit von der Perfektion entfernt ist, hatten wir eine Komplikation mit der FastLED-Bibliothek. Diese setzt globale Variablen fest, auf welcher anscheinend nur in der Main zugegriffen werden darf. Sonst würde ohne jeglichen Grund oder Fehlercode die Bibliothek nicht Funktionieren (die Software stürzt nicht ab, die LEDs leuchten einfach nicht). Dies konnten wir eben durch eine Exportierung in die Main umgehen.

5.3. Geplant

In diesem Abschnitt gehen wir auf die in Zukunft geplanten Teile ein, welche bereits in der Projektskizze erwähnt wurden.

5.3.1. Künstliche Intelligenz

Wir streben nach der Implementierung einer künstlichen Intelligenz, die das alleinige Spielen gegen den Computer, aber auch die Ersetzung aller Spieler durch eine KI, zu Vorführungszwecken ermöglicht. Die Nutzer können dabei genau auswählen wie viele KIs hinzugefügt werden und können somit theoretisch auch zu mehreren gegen eine oder mehrere KIs spielen.

Zur Umsetzung evaluiert die KI die Zugmöglichkeiten basierend auf verschiedenen Kriterien und erstellt eine sortierte Liste mit den besten Zügen. Dabei variieren die Kriterien je nach Spielweise der KI. Es gibt vier verschiedene Spielweisen, die hier näher erläutert werden sollen. Zum einen die aggressive Spielweise, in der die KI versucht Ihr Ziel mit allen Mitteln zu erreichen, unabhängig wie risikobehaftet das Vorgehen ist. Ferner wird noch eine passive Spielweise zur Verfügung gestellt, in der sich die KI zurückhält und möglichst sparsam mit

Tickets umgeht. Und letztlich noch eine taktische Spielart in der die KI die höchste Gewinnrate erzielt.

Die Software wählt dabei den Spielstil zufällig aus, sodass für abwechslungsreiche Spiele gesorgt werden kann. Der Spielstil ist dem Spieler dabei unbekannt und wird ihm lediglich am Ende angezeigt. Der Nutzer bekommt allerdings die Möglichkeit zwischen drei Schwierigkeitsstufen auszuwählen, die bestimmen wie wahrscheinlich es ist, dass auch eine Zugmöglichkeit, die in der Liste weiter unten eingeordnet wurde, gewählt wird. Bei der schwierigsten Einstellung wird dabei immer, der für die Spielweise, beste Zug gewählt.

5.3.2. Neues Interface

Wir möchten dem Spieler ein separates Interface zur Verfügung stellen, sodass dieser, ohne die Voraussetzung ein eigenes internetfähiges Gerät zu besitzen, an dem Spielverlauf teilnehmen kann. Dazu planen wir einen Mikrocontroller, genauer gesagt dem ESP32, als Interface umzufunktionieren. Dieser bekommt hardwaretechnisch zusätzlich drei Buttons und ein LCD, um die Steuerung der virtuellen Oberfläche zu ermöglichen. Softwaretechnisch verbindet sich das Interface als Client mit dem Netzwerk des ESPs und kommuniziert über ein TCP-Interface.

6. Installationsanleitung

Im folgenden Abschnitt wird nun auf die Installation des Spiels eingegangen. Dabei wird auf die Installation für die Weiterentwicklung bzw. der Neuentwicklung (Nachentwicklung) eingegangen.

6.1. IDE und Plugin

Da wir nicht die Arduino IDE genutzt haben und die Arduino IDE auch nicht fähig wäre, dieses Projekt in dem Sinne zu kompilieren, da IDE spezifische Einstellungen genutzt wurden, muss unsere IDE genutzt werden. Wir haben Visual Studio mit dem Plugin Visual Micro verwendet. Visual Studio gibt es öffentlich bei Microsoft kostenlos zum Download und Visual Micro ebenfalls zur Zeit der Fertigstellung des Projektes Kostenlos im Plugin-Bereich von Microsoft. Für beides kann der Link in den Quellen und Referenzen gefunden werden. Für Visual Micro muss Arduino installiert sein.
Um ein ESP32 programmieren zu können muss nun auch das Board ESP32 installiert werden. Dafür installieren wir es in der Arduino IDE, da diese die gleichen Ordner und Dateien wie Visual Micro verwendet. Die komplette Installation vom ESP32 wird hier[1] erklärt. Nachdem der ESP32 installiert wurde nehmen wir folgende Einstellungen vor:

Board: ESP32 Dev Module

1. Upload-Speed: 921600
2. CPU Frequency: 240MHz (WiFi/BT)
3. Flash Frequency: 80MHz

[1] https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md

4. Flash Mode: QIO
5. Flash Size: 4MB (32Mb)
6. Partition Scheme: No OTA (2MB APP/2MB SPIFFS)
7. Core Debug Level: None
8. PSRAM: Disabled

Wichtig noch: Wenn das Projekt als Release kompiliert werden soll, dann muss dieses mit Strg+F5 gestartet werden. Hingegen wird es nur mit F5 als Debug kompiliert.

6.2. Bibliotheken

Üblicherweise werden Bibliotheken in der Arduino IDE unter Tools->Manage Libraries installiert. In dieser Weise müssen folgende Bibliotheken installiert werden:

- LiquedCrystalDisplay_I2C
- FastLED

Wir nutzen zwei Bibliotheken, welche direkt als externe Bibliotheken in Visual Micro importiert werden können. Dazu kann in Visual Studio bei einem als Arduino indexiertes Projekt unter *Erweiterungen>Add Library>Install Library From Zip*, die von GitHub heruntergeladene Bibliothek eingebunden werden.

6.3. Konfiguration

Am Projekt können einige Konfigurationen getätigt werden. Hier werden nun einige wichtige aufgezeigt, wobei deren Bedeutung, deren Ort und der Name in einer Tabelle aufgelistet wird.

Name	Modul	Bedeutung	Standardwert
DEVICE_NET_SSID	Network Manager	Einstellung der SSID des WLANs (Öffentlicher WLAN-Name)	Digital Detectives
DEVICE_NET_PASS	Network Manager	Einstellung des WLAN-Passworts. Muss länger als acht Zeichen sein.	scotyard
DEVICE_NET_LOCAL_IP	Network Manager	Lokale IP innerhalb des WLAN-Netzwerkes, welche die IP des Webservers wird und auf welche auch zugegriffen wird. Muss innerhalb der gegebenen Subnetz-Maske liegen.	{ 192, 168, 0, 1 }
DEVICE_NET_MAX_CONN	Network Manager	Maximale Anzahl der Verbindungen, welche das WLAN maximal akzeptiert. Alle weiteren werden ignoriert. Gibt normalerweise keinen Grund, diese Zahl zu verändern.	6

DEVICE_LCD_ADDRESS	Output Manager	Speziell abhängige LCD-Adresse mit welcher durch den Serielle Kommunikation auf diesen zugegriffen wird.	0x27
DEVICE_MIN_REMAIN_MEMORY	Fault Handler	Minimaler üblicher Speicher, welcher zu Verfügung stehen muss. Wenn nicht, wird das Spiel neugestartet. Dieser Wert sollte nicht zu niedrig sein, da das Erstellen von Webseiten sehr viel Speicher braucht.	50000
DEVICE_MAX_HEAP_FRAGMENT	Fault Handler	Minimale Fragmentierung, welche existieren darf. Angegeben in Prozent.	50
DEVICE_FID_OVERFLOW	Failure Handler	Größtmöglich Fehler ID, welche den Failuremanager erreichen darf. Diese ist so niedrig, da die Zeit des FailureManagers abhängig von der FehlerID ist.	16
DevicePins.h	DevicePins.h	Alle Pins, welche vom ESP32 genutzt wurden. Dabei werden die Makros indirekt referenziert.	siehe DevicePins.h
WEB_MAX_CONN	WebServer Manager	Maximale Anzahl an Nutzern, welche maximal mit dem Webserver verbunden sein können.	10
COMM_WEBINT_WEBSOCK_TIMEOUT	Web Interface	Die Zeit, welche ein Websocket maximal brauchen darf um sich verbunden.	2000
COMM_WEBINT_WEBSOCK_PING_INT	Web Interface	Die Zeit, welche ein Ping maximal für einen Pong brauchen darf.	3000
GAME_BOARDMANAGER_COLLECT_LCDSPEED	Board Manager	Geschwindigkeit in Millisekunden, in welcher sich die Animation im BoardManager während der Collecting-Phase bewegt.	500
GAME_BOARDMANAGER_RUNNING_LEDSPEED_REP	Board Manager	Die Zeit, welche die LED braucht, um einmal aufblinken und wieder aus zu gehen.	1440
GAME_BOARDMANAGER_RUNNING_LEDSPEED	Board Manager	Die Zeit, welche eine LED braucht, um die LED um einen Tick, also eine Helligkeitsstufe, zu erhöhen.	120

7. Bedienungsanleitung

In diesem Abschnitt wird nun das Spiel, im vollen Umfang beschrieben. Dabei wird auf die Eingabe mithilfe der Knöpfe und den LEDs, den Zugriff des Netzwerkes, die Bedienung der Website und auf spezielle weitere Bedingungen eingegangen. Softwaretechnische Konfigurationen und Anpassungen sind unter Installationsanleitung, Konfiguration angegeben.

7.1. Knöpfe und LCD

Die Knöpfe und das LCD gelten als ein öffentliches Eingabemittel, welches bis auf das Starten des Spiels nicht unbedingt Spielrelevant ist.

7.1.1. Einschalten

Das kurze Drücken des mittleren Knopfes schaltet das Gerät ein. Ein zu kurzes Drücken sollte wenn möglich verhindert werden, da dies das Gerät schädigt.

7.1.2. Bedeutung der Knöpfe

Die klassische Bedienung der Knöpfe wird in der folgenden Abbildung dargestellt. Es gibt auch besondere Bedienungsweisen, wie bei Ja / Nein Fragen, welche die Bedienung mithilfe des LCDs anzeigt.

UP ENTER DOWN

7.1.3. Spiel starten

Sobald mindestens 4 Spieler verbunden sind, kann das Spiel gestartet werden, indem das Menü mithilfe des ENTER-Buttons geöffnet, anschließend durch den DOWN-Button zu dem Punkt **Start Game** navigiert und letztlich mit dem ENTER-Button bestätigt wird. Da zwei Phasen zu durchlaufen sind, muss dieser Vorgang möglicherweise wiederholt werden.

7.1.4. Einstellungen

Das Einstellungsmenü kann durch Drücken des ENTER-Buttons geöffnet werden. Je nachdem in welcher Phase sich das Spiels befindet, unterscheiden sich die Menüs.
Navigation in dem Einstellungsmenü kann durch betätigen der drei Tasten (UP, DOWN, ENTER) vollzogen werden. Dabei wird das Spiel, solange das Menü geöffnet ist, pausiert.

Die unterschiedlichen Menüs bieten folgende Möglichkeiten, die später näher erläutert werden:

In der Sammelphase, die vor dem Spielbeginn stattfindet, steht ein Menü zur Verfügung das sowohl die Möglichkeit bietet zu der Einstellungsphase fortzufahren um das Spiel dann starten zu können (Start Game), aber auch die Möglichkeit einen Neustart durchzuführen (Restart), auf das Spielermenü zuzugreifen (Players) und eine KI hinzuzufügen (Add Bot).

In der Einstellungsphase, die direkt nach der Sammelphase stattfindet, gibt es im Menü die Möglichkeit das Spiel zu starten (Start Game), auf das Spielermenü zuzugreifen (Players) und das Board zurückzusetzen (Abort).

In der Spielphase gibt es die Möglichkeit auf das Modifikationsmenü zuzugreifen (Modification), das einem erlaubt Spielern Tickets hinzuzufügen und die Position eines Spielers beliebig zu setzten, um zum Beispiel eine ausversehen falsch angeklickte Zugmöglichkeit rückgängig zu machen. Des Weiteren kann auf das Spielermenü zugegriffen werden (Players), das Spiel neugestartet werden (Restart), ein manueller Speichervorgang durchgeführt werden (Resave), das Board heruntergefahren werden (Shutdown) und Debug-Funktionen verwendet werden (PrintSector), die einem ermöglichen alle Daten aus dem Sektor in Hex-Form auszugeben.

7.1.4.1. Spielermenü (Players)

Im Spielermenü gibt es die Möglichkeit, die Identifikationsnummer des ausgewählten Spielers anzuzeigen (Show PlayerId). Diese ist u.a. wichtig, wenn man als selber Spieler ein Spielstand später auf einem anderen Gerät weiterspielen möchte. Des Weiteren kann man Spieler für die aktuelle Partie sperren (Kick). Letztlich stehen noch Debug-Befehle zur Verfügung, die einem die Möglichkeit geben, das aktuell mit dem Spieler assoziierte Interface anzuzeigen (Interface Type) und die Daten des Spielers in Hex-Form auszugeben (Print Sector).

7.1.5. Weiterspielen

Der Mikrocontroller speichert regelmäßig und automatisch den Spielstand um nach einem Fehler, Stromausfall oder absichtlichem Abbruch weiterspielen zu können.
Sie haben des Weiteren die Möglichkeit, jederzeit das Spiel über die Einstellungen manuell zu speichern.

Zur Resümierung des Spiels können sie nach dem Start des Boards, die automatische Abfrage mit dem entsprechenden Button bestätigen.

Nachdem Sie sich über das Web-Interface verbunden haben, werden Sie, wenn Sie sich mit demselben Gerät verbunden haben, und keine Cookies gelöscht haben, automatisch wiederverbunden.

Sollten Sie das Gerät gewechselt haben oder aus anderen Gründen den Cookie nicht mehr besitzen, werden Sie aufgefordert den vorherigen Spieler auszuwählen. Sie sollten dabei ihre vorherige Player-ID oder Farbe kennen.

7.2. Netzwerk

7.2.1. Verbinden als Spieler

Verbinden Sie sich mit dem WLAN-Netzwerk mit der SSID **Digital Detectives.** Das Passwort lautet **scotyard**. Öffnen Sie anschließend einen Browser und navigieren Sie zu der Adresse **http://192.168.0.1**.

7.3. Website

7.3.1. Während des Spiels

Als Spieler bekommen Sie während des Spielvorgangs auf dem Web-Interface sowohl Informationen über Ihre Ticketanzahl als auch die Möglichkeit konkrete Züge auszuführen.

7.3.1.1. Als Detective

Sie können Ihren Zug über das klicken/tippen auf die entsprechenden, generierten Buttons durchführen. Die farbliche Kodierung entspricht dabei den auf dem Board gekennzeichneten Wegen. Es werden dabei nur mögliche Züge angezeigt.

7.3.1.2. Als Villain/Gegner

Sie können Ihre aktuelle Position durch das klicken/tippen auf den Hide-Button verbergen, sodass Sie nicht den Bildschirm des Gerätes verdecken müssen. Das Double-Ticket können Sie nutzen indem Sie die Checkbox ‘Use Double-Ticket’ vor dem Betätigen des Zug-Buttons anklicken. Die Zugmöglichkeiten in der schwarzen Box, entsprechen dabei dem anonymen Zug.

8. Quellen und Referenzen

In diesem Abschnitt werden finale noch einige wichtige Quellen gelistet. Dabei haben wir versucht nur wichtige Quellen für eine mögliche zukünftige Entwicklung auszuwählen.

[1] Scotland Yard Game Rules:
https://www.ultraboardgames.com/scotland-yard/game-rules.php
[2] Scotland Yard (Spiel):
https://de.wikipedia.org/wiki/Scotland_Yard_(Spiel)
[3] Klassisches Scotland Yard Brettspiel
[4] The complexity of Scotland Yard:
https://www.illc.uva.nl/Research/Publications/Reports/PP-2006-18.text.pdf
[5] ESP32 to go:
https://www.heise.de/developer/artikel/ESP32-to-go-4452689.html
[6] Startpositionen der Spieler
https://boardgamegeek.com/thread/77052/starting-places
[7] Näheres zu PROGMEM
https://arduino-esp8266.readthedocs.io/en/latest/PROGMEM.html
[8] ESP8266 / 32 WIFI
https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html
[9] ESP8266 / 32 SoftAP
https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/soft-access-point-class.html
[10] ESP8266WebServer
https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer
[11] Visual Micro
https://www.visualmicro.com/
[12] ESP32 Github
https://github.com/espressif/arduino-esp32
[13] ESP32AsyncWebServer
https://github.com/me-no-dev/ESPAsyncWebServer
[14] AsyncTCP
https://github.com/me-no-dev/AsyncTCP
[15] ESP Fatal Errors
https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/fatal-errors.html
[16] ESP Panic
https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/kconfig.html#config-esp32-panic
[17] ESP Core Dump
https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/core_dump.html
[18] ESP Tools
https://github.com/espressif/esp-idf
[19] ESP AZ-Delivery
https://www.az-delivery.de/products/esp32-developmentboard
[20] ESP AZ-Delivery pinout

https://cdn.shopify.com/s/files/1/1509/1638/files/ESP_-_32_NodeMCU_Developmentboard_Pinout_Diagram.jpg?4479111012146266271
[21] ESP Exception decoder
https://github.com/me-no-dev/EspExceptionDecoder
[22] Visual Studio download
https://visualstudio.microsoft.com/de/downloads/
[23] FastLED
https://github.com/FastLED/FastLED
[24] LiquidCrystalDisplay_I2C
https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library