



Universität Kassel

Fachbereich 16 - Elektrotechnik/Informatik

Gestaltung einer spezialisierten Hardware und der zugehörigen Kernelmodule

für

Multi Player Multi Goal Spiele

Projektarbeit

vorgelegt von: Schweinsberg, Patrick

Abgabedatum: 20. März 2018

Matrikelnummer: 27212826

Lehrveranstaltungs-Nr.: 227013

Studiengang: Informatik

Semester: WS 2017/2018

Inhaltsverzeichnis

1 Einleitung.....	9
1.1 Softwareziel.....	9
1.2 Hardwareziel.....	10
2 Hardwareentwurf.....	12
2.1 Grundlage.....	12
2.2 Beschreibung der Komponenten.....	14
2.2.1 Adapter zum Raspberry Pi.....	14
2.2.2 Game-Pad.....	15
2.3 Belegung der Busschnittstelle.....	15
2.4 Reproduzierbarkeit.....	16
3 Schaltplan.....	18
3.1 Port-Adapter.....	18
3.2 Game-Pad.....	20
4 Layout.....	22
4.1 Port-Adapter.....	22
4.2 Controller-Pad.....	23
4.3 Herstellung.....	24
4.3.1 Fräsen.....	24
4.3.2 Handbestückung.....	24
4.3.3 Prototypen.....	25
5 Erster Test.....	28
5.1 Vorbereiten des Raspberry Pi und Test der Betriebsbereitschaft.....	28
5.2 Test des MCP23017 mit i2ctools.....	28
5.3 Testen des Displays.....	30
6 Treiberentwicklung.....	31
6.1 Kernel-Header installieren.....	31
6.2 Konzepte.....	32
6.2.1 Port-Adapter.....	33
6.2.2 Game-Pad.....	34
6.2.3 TFT-Display.....	34
6.3 Zugriff auf MCP23017.....	35
6.4 Gameport Schnittstelle.....	39
6.4.1 Platform_Data zum Informationsaustausch.....	41
6.4.2 Variablen mit static Deklaration.....	42
6.5 Framebuffer Schnittstelle.....	43
6.6 Implementierung.....	43
6.6.1 Implementierung des Gameport-Treibers „mpmrgg-port“.....	43
6.6.2 Implementierung des Game-Pad-Treibers „mpmrgg-pad“.....	44
6.6.3 Implementierung des Display-Treibers „mpmrgg-st7735“.....	46
6.7 Anpassungen nach dem Test mit mehreren Game-Pads.....	50
6.7.1 Parallele Initialisierung der Displays.....	50
6.7.2 Anpassungen zur Verminderung von Bitfehlern auf dem I ² C-Bus.....	52
7 Test des Treibers.....	53
7.1 Test des Adapters.....	53
7.2 Test des Game-Pads.....	53

7.3 Test des Framebuffers.....	54
8 Automatismen.....	56
8.1 Automatisches Laden der Treiberstruktur beim Starten des Kernels.....	56
9 Weitere Aufgaben im Projekt.....	57
9.1 Aufbau der Entwicklungshardware auf Basis des Raspberry Pi.....	57
9.2 Konfiguration und Wartung des Raspberry Pi.....	58
9.3 Grafiken und Leveldesign für das Spiel.....	59
9.3.1 Grafiken für das Spiel.....	60
9.3.2 Leveldesign des Spiels.....	66
9.4 Umzug auf das finale Betriebssystem.....	86
9.5 Einbinden des Spiels in die Emulationstation Umgebung.....	86
9.6 Automatisches Setzen des Logos auf den Game-Pads.....	87
10 Ausblick.....	89
Anhang.....	91
Literaturverzeichnis.....	92
A Bücher/Monographien.....	92
B Internet-Adressen.....	92

Abkürzungsverzeichnis

MPMGG	Multi Player Multi Goal Game
SPI	Serial Peripheral Interface
SNES	Super Nintendo Entertainment System
NES	Nintendo Entertainment System
RPI3	Raspberry Pi 3
TFT	Thin-film transistor-Display
GCC	GNU Compiler Collection
DRAM	Display Random-Access Memory
GPIO	general purpose input/output
I ² C	Inter-Integrated Circuit
MHz	Mega Hertz
GB	Giga Byte
GHz	Giga Hertz
OLED	organic light emitting diode
LED	Light emitting diode
MBps	Mega Bits pro Sekunde
kBps	Kilo Bits pro Sekunde
ssh	secure shell
GPU	Graphics processing unit

Abbildungsverzeichnis

Abbildung 1.1: Blockdiagramm des Aufbaus.....	11
Abbildung 2.1: Raspberry Pi 3 mit Funktionen am Konnektor.....	12
Abbildung 2.2: TFT mit ST7735 Prozessor.....	13
Abbildung 2.3: Step Down Konverter.....	14
Abbildung 2.4: Busbelegung der 2x5 IDC-Stiftleiste.....	16
Abbildung 2.5: Busbelegung der 1x10 Stiftleiste.....	16
Abbildung 2.6: Ätzbereit vorbereitetes Layout auf Platinen-Basismaterial.....	17
Abbildung 3.1: Raspberry Pi 3 GPIO Header.....	18
Abbildung 3.2: Schaltplan zum Port-Adapter.....	19
Abbildung 3.3: Schaltplan zum Game-Pad.....	21
Abbildung 4.1: Layout zum Port-Adapter.....	22
Abbildung 4.2: Layout zum Controller-Pad.....	23
Abbildung 4.3: Prototyp des Port-Adapters.....	25
Abbildung 4.4: Port-Adapter mit den nötigen Schnitten.....	26
Abbildung 4.5: Prototyp des Game-Pad ohne Display.....	26
Abbildung 4.6: Prototyp des Game-Pad (montiert).....	27
Abbildung 4.7: Prototyp des Game-Pad, Rückseite.....	27
Abbildung 6.1: Auszug aus dem Datenblatt des MCP23017.....	36
Abbildung 6.2: Treiberebenen für die Geräte.....	42
Abbildung 6.3: Tastenbelegung des Game-Pads.....	45
Abbildung 6.4: 16-Bit Farbaufteilung pro Pixel.....	49
Abbildung 6.5: I ² C-Adressbyte des MCP23017.....	51
Abbildung 9.1: Rückseite des Raspberry Pi Touch Display mit montiertem RPI.....	57
Abbildung 9.2: Finale Konfiguration.....	58
Abbildung 9.3: raspi-config.....	59
Abbildung 9.4: Einzelbilder einer Spriteanimation.....	60
Abbildung 9.5: d-man_work.png.....	60
Abbildung 9.6: actors.png.....	61
Abbildung 9.7: objects.png.....	62
Abbildung 9.8: hud.png.....	62
Abbildung 9.9: menu.png.....	63
Abbildung 9.10: tileset-oldstyle.png.....	63

Abbildung 9.11: tileset-chain.png.....	64
Abbildung 9.12: tileset-forest.png.....	64
Abbildung 9.13: tileset-warehouse.png.....	65
Abbildung 9.14: tileset-desert.png.....	65
Abbildung 9.15: tileset-ship.png.....	66
Abbildung 9.16: Übersicht aller Ebenen.....	67
Abbildung 9.17: Objektebene.....	67
Abbildung 9.18: Vordergrund.....	68
Abbildung 9.19: Hintergrund.....	68
Abbildung 9.20: Hauptebene.....	69
Abbildung 9.21: Übersicht aller Ebenen.....	69
Abbildung 9.22: Objektebene.....	70
Abbildung 9.23: Vordergrund.....	70
Abbildung 9.24: Hintergrund.....	71
Abbildung 9.25: Hauptebene.....	71
Abbildung 9.26: Übersicht aller Ebenen.....	72
Abbildung 9.27: Objektebene.....	73
Abbildung 9.28: Vordergrund.....	74
Abbildung 9.29: Hintergrund.....	75
Abbildung 9.30: Hauptebene.....	76
Abbildung 9.31: Übersicht aller Ebenen.....	77
Abbildung 9.32: Objektebene.....	77
Abbildung 9.33: Vordergrund.....	77
Abbildung 9.34: Hintergrund.....	78
Abbildung 9.35: Hauptebene.....	78
Abbildung 9.36: Übersicht aller Ebenen.....	79
Abbildung 9.37: Objektebene.....	79
Abbildung 9.38: Vordergrund.....	80
Abbildung 9.39: Hintergrund.....	80
Abbildung 9.40: Hauptebene.....	81
Abbildung 9.41: Übersicht aller Ebenen.....	81
Abbildung 9.42: Objektebene.....	82
Abbildung 9.43: Vordergrund.....	82

Abbildung 9.44: Hintergrund.....	83
Abbildung 9.45: Hauptebene.....	83
Abbildung 9.46: Übersicht aller Ebenen.....	84
Abbildung 9.47: Objektebene.....	84
Abbildung 9.48: Vordergrund.....	84
Abbildung 9.49: Hintergrund.....	84
Abbildung 9.50: Hauptebene.....	85
Abbildung 9.51: Übersicht aller Ebenen.....	85
Abbildung 9.52: Objektebene.....	85
Abbildung 9.53: Vordergrund.....	85
Abbildung 9.54: Hintergrund.....	85
Abbildung 9.55: Hauptebene.....	86

Tabellenverzeichnis

Tabelle 1.1: Vergleich bestehender Systeme.....	10
Tabelle 6.1: Registerkonfiguration des MCP23017.....	38
Tabelle 6.2: Adresszuordnung am I ² C-Bus mittels Codierschalter.....	40
Tabelle 6.3: Tastenbelegung.....	45
Tabelle 6.4: Registerkonfiguration des ST7735.....	46
Tabelle 6.5: Dateizugriffe des „mpmgg-st7735“-Treibers.....	48
Tabelle 6.6: Einstellungen zur Auflösung des „mpmgg-st7735“-Treibers.....	49
Tabelle 6.7: Einstellungen zur Auflösung des „mpmgg-st7735“-Treibers.....	50
Tabelle 7.1: Verwendete Parameter des fbi-Kommandos.....	55

1 Einleitung

Diese Projektarbeit befasst sich mit dem Bau eines Prototypen und der Erstellung spezialisierter Softwarekomponenten für die Erfüllung der Hardwareanforderungen. Auf Basis eines Raspberry Pi 3 Modell B wird ein Spiel entwickelt welches für den Betrieb ein besonderes Interface erfordert. Das Projekt wird von zwei Studenten bearbeitet, wobei die Spielentwicklung betreffenden Anforderungen von *Andreas Nöding* bearbeitet werden. Alle Hardware betreffenden Aufgaben werden von *Patrick Schweinsberg* bearbeitet. Zusätzlich sind Level- und Grafikdesign an *Patrick Schweinsberg* ausgelagert.

Das Spiel wird in Abschnitt **1.1 Softwareziel** genauer beschrieben. Auf die hierfür benötigte Hardware wird in Abschnitt **1.2 Hardwareziel** genauer eingegangen. Ziel dieser Arbeit ist die Dokumentation der in Abschnitt 1.2 beschriebenen Zielsetzung bezüglich der Hardwareanforderungen, bzw. der Dokumentation in Bereich Level- und Grafikdesign geleisteten Arbeiten. Im Anhang befindet sich Datenblätter, so wie eine Kurzdokumentation zum eigentlichem Spiel. Auf der zugehörigen DVD ist der finale Entwicklungsstand in Source-Daten, so wie das Image für den Raspberry Pi 3 mit allen ausführbaren Daten zu finden.

1.1 Softwareziel

Die Idee hinter dem Spiel basiert auf Geheimhaltung. Jeder Spieler kann Attribute, Ausrüstung und Ziele besitzen die vor den anderen Mitspielern geheim zu halten sind. Als Namen für dieses Spielprinzip wurde „multiplayer, multigoal gaming“ (MPMGG) gewählt. Als Rahmen der eigentlichen Spielhandlung wurde für das MPMGG-Spiel ein Arena-Shooter gewählt. Für das Konzept bietet dieser Spieltyp eine ausreichende Bühne; auf eine packende Spielgeschichte muss kein Wert gelegt werden, das Erarbeiten neuartiger Konzepte und die technische Umsetzung steht im Mittelpunkt.

Die Spieler bekommen ihre Eigenschaften diskret mitgeteilt, der Austausch dieser Information ist nun den Spielern überlassen. Das Spiel gibt keine Anleitung wie sich zu verhalten ist und jeder Spieler muss entscheiden was ihm dienlich erscheint.

Wichtig für ein ausgewogenes Spiel ist das Verhalten und die Fähigkeiten der Figuren so zu gestalten, dass das Verhalten überzeugend realistisch und glaubwürdig oder wie von anderen ähnlichen Spielen bereits bekannt erscheinen.

Eine gewisse Geräuschkulisse unterstützt den Eindruck und ist unabdingbar für eine intensive Spielerfahrung.

Die Gestaltung des Spiels soll die begrenzte Leistung des Raspberry Pi 3 nicht mit 3D-Anwendungen oder der Simulation einer komplexen Umgebungssimulation voll auslasten. Zudem soll der Charme früherer erfolgreicher Spiele konserviert werden. Die Grafik ist daher an alte Spiele aus der Ära des Nintendo Entertainment System (NES), des Super Nintendo Entertainment System (SNES) und des Sega Mega Drive angelehnt. Spiele wie aus den Mario und Megaman Serien dienen dabei als Vorbild. Ziel ist eine Retrografik mit einfachen Bildwechselanimationen. Erkennbare Pixel sind ein gewollter Nebeneffekt der Retrografik.

1.2 Hardwareziel

Zur Steuerung des Spiels soll ein Controller verwendet werden, welcher an klassische Konsolen angelehnt ist. Das eigentliche Spiel soll auf einem Fernseher abgebildet werden. Als Unterschied zu auf dem Markt bestehenden Konsolen stehen die deutlich geringere Leistung des Raspberry Pi 3 Modell B (RPI3), die deutlich geringere Energieaufnahme und die Anforderung dem Spieler seine Informationen privat mit zu teilen. Tabelle 1.1 gibt einen Überblick über die Leistung der derzeit Marktführenden Konsolen im Bezug zum Raspberry Pi [I3].

Tabelle 1.1: Vergleich bestehender Systeme

Name	Arbeitsspeicher	Prozessor	Grafikprozessor:	Leistungs-aufnahme
Pi 3 Modell B [©]	SDRAM 1 GB	BCM2837 4 x 1.2GHz	Broadcom Dual Core VideoCore IV, 400MHz	Max. 4 W/h
Play Station 4 [©]	GDDR5 8 GB	Benutzerdefinierter Einchip-Prozessor: 8-Core x86-64 AMD "Jaguar" 8 x 1.6 GHz	AMD Radeon™-Grafik- Engine, 800 MHz	Max. 165 W/h
XBOX One [©]	DDR3 8 GB	AMD Jaguar 8 x 1.75 GHz	AMD Graphics Core Next, 768 Shader, 16 ROPs, 853 MHz	Max. 120 W/h
Nintendo Wii U [©]	2 GB	PowerPC 3 x 1.2 GHz	AMD GPU7, 550 MHz	Max. 30 W/h

Selbstverständlich versucht diese Projektarbeit nicht den Anspruch zu erfüllen eine konkurrenzfähige Konsole zu produzieren. Viel mehr dient diese Projektarbeit dem Kennenlernen bestehender Schnittstellen des Linux Betriebssystems und der Entwicklung auf einer limitierten Hardware, wie sie auch in eingebetteten Systemen gebräuchlich ist.

Die Forderung dem Spieler private Informationen zukommen zu lassen, führte zu der Idee jeden Controller um ein kleines Display zu ergänzen. Die Tasten der Controller sollen via Port-Expander vom Betriebssystem abgefragt werden. Die Allzweckeingabe/-Ausgabe (engl. general purpose input/output, kurz GPIO) Pins des Raspberry Pi sollen nicht separat pro Controller belegt werden. Es soll ein einheitlicher Bus geschaffen werden, welcher parallele Verdrahtung der Controller und der Komponenten ermöglicht.

Die Taster der Controller, so wie die Steuerleitungen des Displays werden direkt mit dem MCP23017 verbunden. Der adressierbare MCP23017 Schaltkreis, hergestellt von Microchip Technology Inc. stellt als Port-Expander das Herzstück der Controller dar. Der MCP23017 bietet 16 konfigurierbare Ein- oder Ausgänge, eine I²C-Schnittstelle zur Kommunikation mit dem Bus-Master, drei Anschlusspins zur Konfiguration der Adresse am verwendeten Inter-Integrated Circuit-Bus (I²C-Bus) und ist in verschiedenen Gehäusetypen verfügbar.

Der verwendete Display ist ein 1,8 Zoll Thin-film transistor-Display (TFT) mit einem integrierten ST7735 Schaltkreis welcher zur Ansteuerung dient. Dieses TFT-Modul benötigt neben den beiden Anschlusspins für das serielle Peripherie-Interface (SPI) drei weitere Anschlusspins zur Steuerung, sowie eine 3.3Volt Versorgung.

Abbildung 1.1 zeigt ein Blockdiagramm welches das Zusammenspiel der einzelnen Komponenten vom Game-Pad bis zum RPI3 verdeutlicht.

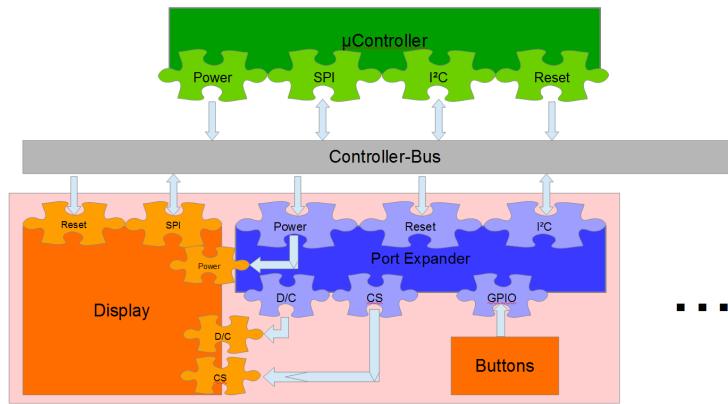


Abbildung 1.1: Blockdiagramm des Aufbaus

Als Schnittstelle des Spiels zu dieser Peripherie dienen die Betriebssystemschnittstellen des zugrundeliegenden Linux-Kernels. Die Game-Pads werden von spezieller Software bedient, diese wird als Kernelmodul bezeichnet und dient als Interface zwischen den vom Kernel spezifizierten Schnittstellen und der Peripherie.

Ziel dieses Hardwareentwurfs ist es zum einen die Anforderungen des Spiels zu erfüllen, so wie die Einarbeitung in die Thematik Schaltungsentwurf, Layout und Erstellung von Treibern für Linux-Betriebssysteme unter Verwendung einer Komplexen Peripherie. Das Grundlagenwissen zur Treiberentwicklung mit der Programmiersprache C, sowie über Speicher-Allokation, der Schnittstellenbeschreibungen und der Methodik des allgemeinen Treiberentwurfs werden um ein tieferes Verständnis der verwendeten Schnittstellen wie GPIO-Zugriff, des SPI, des I²C-Bus, des Aufbaus und der Verwendung des Joystick-Interfaces, so wie des Framebuffer-Interfaces erweitert. Die Schnittstelle des Joystick- und des Framebuffer-Interfaces werden vom Betriebssystem zur Verfügung gestellt. Die benötigten Arbeitsschritte zum Erreichen dieser Ziele werden in dieser Dokumentation behandelt und beschrieben.

2 Hardwareentwurf

Zu Beginn der Entwicklung erfolgte die Erstellung des benötigten Hardware-Prototypen. Die folgenden Abschnitte dienen zur Dokumentation dieses Entwicklungsabschnittes. Die hierfür getroffenen Entscheidungen werden verdeutlicht und die auftretenden Probleme sowie ihre Lösungen betrachtet.

2.1 Grundlage

Dieser Abschnitt beschreibt fertige Hardware Komponenten welche genutzt, allerdings nicht im Rahmen dieses Projektes entwickelt wurden.

Als bestehende Plattform wird der Raspberry Pi 3 Modell B verwendet. Dieses Mikrocontroller-Entwicklungsboard ist weit verbreitet, mit einem Preis von ca. 37 Euro kostengünstig und bietet mit den vier Kernen auf 1200 Mega Hertz (MHz) des BCM2837 Prozessors und einem Giga Byte (GB) Arbeitsspeicher genügend Leistung für die bevorstehende Aufgabenstellung. Das Board verfügt unter anderem über eine HDMI-Schnittstelle für ein Fernsehgerät, einer Netzwerkschnittstelle, einem Cardreader für Micro-SD-Karten von der auch das Betriebssystem geladen wird, vier USB-Schnittstellen und einen Konnektor um die Schnittstellen und GPIOs des Prozessors direkt zu nutzen. Zudem sind bereits diverse angepasste Linuxkernel, Frontends, so wie eine Breite Masse von diversen Bibliotheken und Frameworks verfügbar. Da sich das Betriebssystem, sowie alle Softwarekomponenten des Systems auf der Micro-SD-Karte befinden lässt sich der Speicher leicht erweitern. Für die Entwicklung wurde ein 16 GB Speicher eingesetzt. Abbildung 2.1 aus Quelle [I1] zeigt den Raspberry Pi mit einer Beschreibung der am Konnektor abgreifbaren Pins.

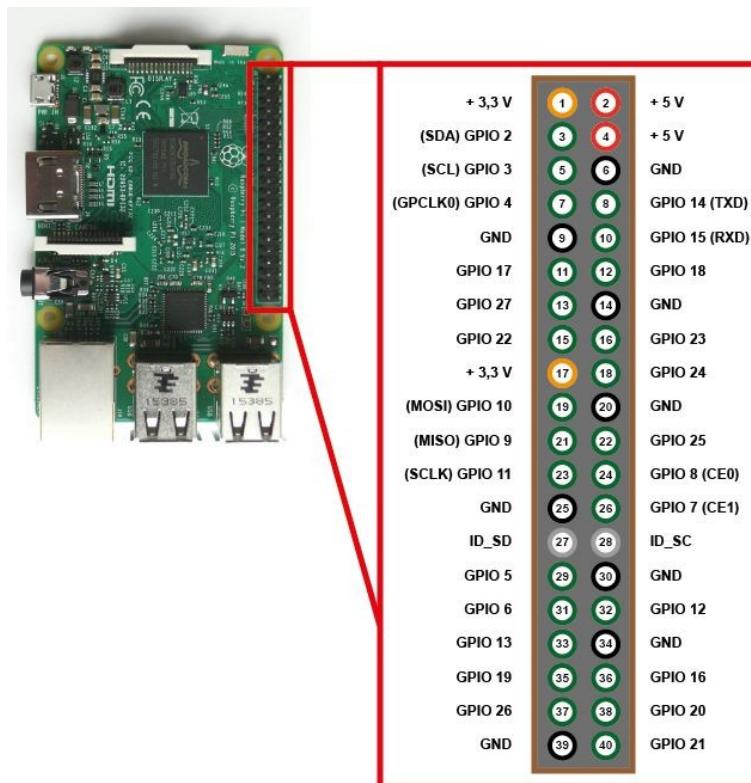


Abbildung 2.1: Raspberry Pi 3 mit Funktionen am Konnektor

Weitere bereits bestehende Komponenten sind die TFT-Displays mit ST7735 Prozessor. Abbildung 2.2 zeigt das verwendete Display [I2]. Es handelt sich um 1,8 Zoll TFT auf Basis der organischen Leuchtdioden (englisch organic light emitting diode, kurz OLED) Technologie mit einer Auflösung von 128 x 160 Bildpunkten und 16 Bit Farben. Zum Betrieb wird eine 3.3 Volt Stromversorgung benötigt. Neben der optionalen Steuerung der Hintergrundbeleuchtung (BL) und den beiden Pins für die Stromversorgung (GND, VCC), verfügt das Display über einen Chipselect- (CS), einen Daten/Controll- (DC), einen Takt- (SCL) und einen Daten-Eingang (SDA) für die Verwendung des SPI.



Abbildung 2.2: TFT mit ST7735 Prozessor

Zur Energieversorgung des Raspberry Pi wird ein Netzteil benötigt welches 5 Volt über einen Micro-USB-Anschluss liefert. Die Peripherie der Game-Pads benötigt eine 3.3 Volt Stromversorgung, diese wird durch die Bus-Schnittstelle an die Verbraucher geliefert. Der Raspberry Pi bietet eine eigene 3.3 Volt Versorgung, jedoch würde der potentielle Bedarf der Game-Pads diese maximal ausreizen, dies könnte eine Instabilität des Prozessors zur Folge haben. Um dies zu vermeiden wird die Spezifikation des Adapters in Version 1.1 um einen günstig erworbenen Step Down Konverter, welcher auf 3.3 Volt fest trimmbar ist für die Versorgung der Game-Pads erweitert. Er liefert maximal 3 Ampere im Dauerbetrieb. Abbildung 2.3 aus [I4] zeigt einen Auszug der „Betriebsanleitung“ des verwendeten Bauteils. Ein offizielles Datenblatt kann leider nicht zur Verfügung gestellt werden. Die entsprechenden Bezugsquellen stellen die Informationen zu dem Bauteil lediglich als Bildmaterial zur Verfügung.

Instructions



Abbildung 2.3: Step Down Konverter

2.2 Beschreibung der Komponenten

In den folgenden Abschnitten wird die entwickelte Hardware vorgestellt und auf Besonderheiten und zu beachtende Effekte bei der Entwicklung hingewiesen. Dieses Kapitel der Dokumentation befasst sich nur mit der generellen Verwendung und Funktionsweise der Komponenten. Detailliertere Beschreibungen folgen.

2.2.1 Adapter zum Raspberry Pi

Um die Pins des Raspberry Pi in einer vereinheitlichten Bus-Schnittstelle nutzbar zu machen wird eine Aufsteckplatine verwendet, welche die benötigten Pins als Parallel-Bus verfügbar macht. Ebenso auf dieser Platine befindet sich ab Version 1.1 die Stromversorgung des RPI3. Diese besteht aus einer 2,5 mm Hohlbuchse und einem Steckplatz für den Step Down Konverter. Der Raspberry Pi wird über die Hohlbuchse versorgt, wodurch sich die Anforderung an das Netzteil mit Hohlbuchsenstecker auf 5 Volt bei 4 Ampere ergibt.

Für den Bus an dem die Game-Pads angeschlossen werden, müssen der I²C-Bus, ein GPIO, die 3,3 Volt Stromversorgung und der SPI-Bus mit den Pins der zehn poligen Bus-Anschlüssen verbunden werden.

Zu beachten ist das der Raspberry Pi am I²C eingehende Signale verarbeiten können muss. Diese können allerdings nur dann korrekt gelesen werden wenn sie einen maximalen Pegel von 3,3 Volt verfügen. Höhere Signalpegel werden durch die Schutzdioden des Raspberry Pi abgebaut.[I13] Dies sorgt dafür, das Signale falsch vom Bus gelesen werden. Erkennbar ist dies daran das nur noch 0xFF vom Bus gelesen werden kann. Ein vorheriger Versuch den Step Down Konverter ein zu sparen und dafür

die Game-Pads mit 5 V zu betreiben schlug wegen dieser Konstellation fehl.

2.2.2 Game-Pad

Die einzelnen Game-Pads teilen sich einen gemeinsamen Kommunikationsbus von dem sie den benötigten I²C-Bus, einen GPIO für den Reset aller Komponenten am Bus und den SPI-Bus beziehen. Zur Teilnahme am Bus wird die Infrastruktur des MCP23017 verwendet. Dieser Schaltkreis wird als IO-Kontroller verwendet, er schreibt die Steuersignale des TFT-Moduls ebenso wie er die Inputs der Taster liest. Die integrierte Adressierung dieser Schaltung wird somit auch zur Adressierung des Displays verwendet.

Die Adressierung des Port-Expanders geschieht mittels drei Eingangssignalen am MCP23017. Die Adressierung kann durch den verwendeten Adressencoder SRT-10c eingestellt werden. Dieses Bauteil kann die Zahlen 0 bis 10 binär codieren. Die Stellungen welche das höchstwertige vierte Bit belegen werden von einer Licht emittierende Diode (LED) signalisiert. Die Überschreitung der maximalen Adresse wird dabei nicht weiter beachtet und löst keine weiteren Aktionen aus.

Bei der Wahl der Taster wurden geräuschfreie Taster verwendet. Ähnlich der meisten Controller bestehen diese aus einem Gumminippel welcher sanft den Kontakt schließt. Es wurde darauf geachtet, dass diese Taster auch ohne Gehäuse in sich stabil sind und nicht auf der Kontaktstelle verrutschen können.

Zur Visualisierung der persönlichen Informationen fiel die Wahl auf ein 1,8 Zoll TFT-Displaymodul. Dieses wird am Game-Pad befestigt. Es sollte dabei nicht als herausstehendes Teil die Haptik des Game-Pad stören. Das gewählte Displaymodul bietet 16 Bit Farbtiefe auf 128 x 160 Pixel. Der Controller des Displaymoduls ist ein ST7735 Prozessor.

2.3 Belegung der Busschnittstelle

Der Bus welcher die einzelnen Game-Pads verbindet besteht aus sieben Einzelsignalen. Bedingt durch die verwendete Hardware werden die Inter-Integrated Circuit (I²C) Schnittstelle zur Kommunikation mit dem MCP23017, der SPI-Bus zur Kommunikation mit dem ST7735 Prozessor des Displaymoduls und die von allen MCP23017 und ST7735 geteilten Reset-, 3v3- und GND-Verbindung. Als physikalisches Medium zur Übertragung der Informationen dient ein 10 poliges Flachbandkabel welches auf der Game-Pad-Seite mit einer Stifteleiste verbunden oder fest verlötet wird. Auf der Port-Seite wird eine 2x5 IDC-Stifteleiste verwendet um die Game-Pads austauschbar an den Raspberry Pi 3 zu verbinden.

Die Struktur des Bus sorgt dafür, dass jeder Busteilnehmer an jedem beliebigen Steckplatz angeschlossen werden kann. Wichtig ist hierbei nur zu beachten, dass jede Adresse am Bus nur ein einziges mal vorkommt.

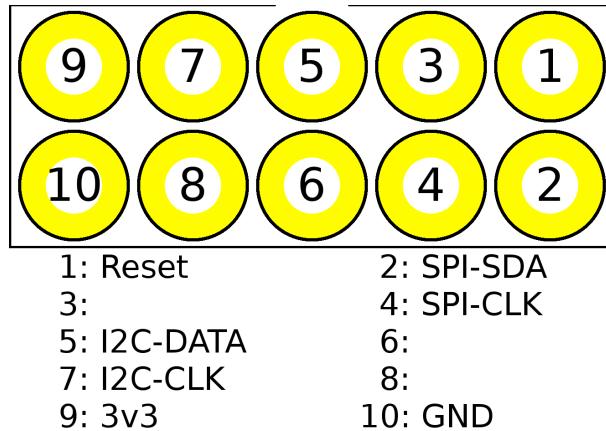


Abbildung 2.4: Busbelegung der 2x5 IDC-Stiftleiste

Abbildung 2.4 zeigt die Belegung der 2x5 IDC-Stiftleiste. Diese ist maßgeblich durch das Layout des Game-Pad beeinflusst. Durch die Kammstruktur der Aufsteckplatine (im später auch Port genannt) sind an dieser Stelle einfache Anpassungen der Verdrahtung möglich. Auf der Platine des Game-Pad ist der Platz jedoch stärker begrenzt und die Möglichkeiten werden durch den Anspruch der einfachen Reproduzierbarkeit weiter eingeschränkt.

An dieser Stelle mag ein Kritikpunkt des derzeitigen Layout genannt werden. Die Adresse der Game-Pads werden von Benutzer am Game-Pad selbst eingestellt. Dieser Umstand kann zu Fehlern führen. Da die MCP23017 Schaltungen 3 Bits zur Konfiguration der Adresse besitzen und am derzeit verwendeten Übertragungsmedium noch 3 Leitungen zur Verfügung stehen, könnten diese zur Adressierung der Game-Pads verwendet werden, das Bauteil SRT-10c kann somit eingespart werden. Die Adresse eines jeden Game-Pads würde bei dieser Konfiguration von der am Port verwendeten 2x5 IDC-Stiftleiste abhängen. Nachteil dieser Konfiguration wäre die steigende Komplexität der Verdrahtung im Layout des Game-Pads.

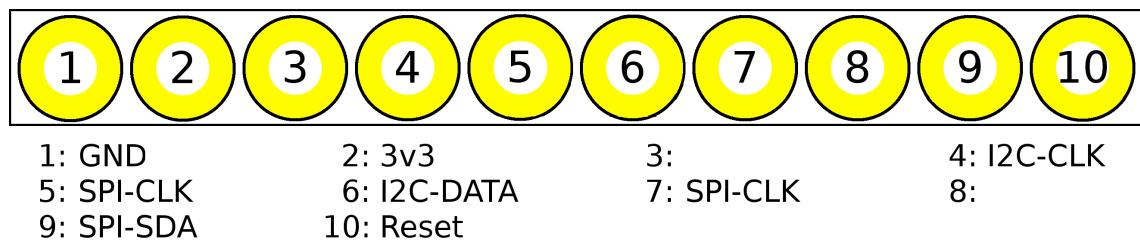


Abbildung 2.5: Busbelegung der 1x10 Stiftleiste

Abbildung 2.5 zeigt die Belegung der 10 poligen Busschnittstelle auf Seite des Game-Pads. Diese Schnittstelle liefert alle Versorgungs- und Funktionsleitungen der Game-Pads und der zugehörigen Displays. Es wurde auf eine volle Parallelisierbarkeit der Busteilnehmer und die Minimalisierung der benötigten Leitungen geachtet.

2.4 Reproduzierbarkeit

Das Ziel dieses Projekts ist die nicht die Produktion einer marktfähigen und fertigen Spielekonsole. Es wird vielmehr die Umsetzung einer Konzeptidee beabsichtigt. Die weitere Verwendung der Ergebnisse steht insofern nicht fest.

Eine mögliche Idee zur weiteren Verwendung der in dieser Projektarbeit erarbeiteten Ergebnisse ist die Veröffentlichung in einem OpenSource-Projekt. Zu diesem Zweck wurde vorausgesetzt das die Reproduktion der benötigten Bauteile mit sehr einfachen Mitteln auch ohne, dass spezialisierte Software und Maschinen möglich sein sollte. Aus diesem Grund wurde mit dem Tonertransferdruck experimentiert. Diese Methode ermöglicht es unter Zuhilfenahme eines Laserdruckers und eines Bügeleisens, Platinen für einen Ätzvorang vor zu bereiten.

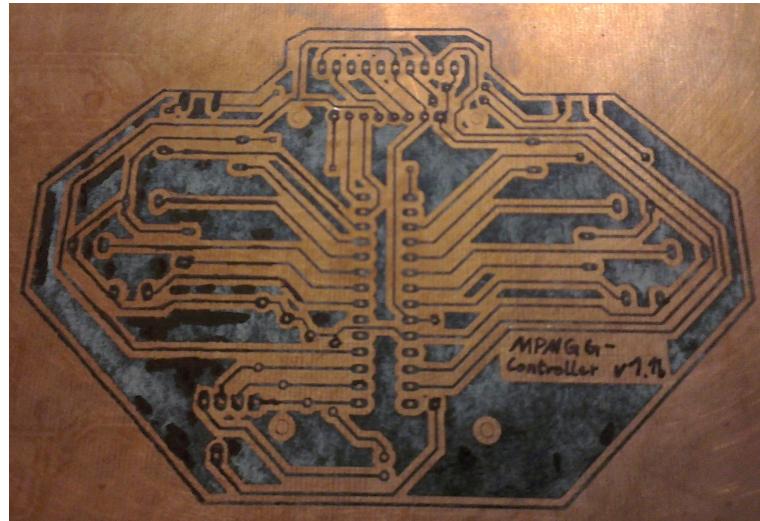


Abbildung 2.6: Ätzbereit vorbereitetes Layout auf Platinen-Basismaterial

Abbildung 2.6 zeigt das Ergebnis eines mittels dieser Methode vorbereiteten Bottom-Layer auf einem Platinen-Basismaterial ohne Fotobeschichtung mit 35µm Kupferbeschichtung. Aus diesem Grund wurden im späteren Layout extra breite Leiterbahnen verwendet. Diese lassen sich besser auf das Rohmaterial übertragen. Die Anzahl der Durchkontaktierungen (auch Vias genannt) wurde auf ein Minimum beschränkt. Zudem liegen alle Vias in einer geraden Linie. Da mittels Tonertransferdruck nur Single-Side-Platinen sehr einfach hergestellt werden können kann der Top-Layer der Platine auf diese Weise durch einfache Drahtbrücken ersetzt werden, was nur das Vorbereiten des Bottom-Layers erfordert. Auch bei den verwendeten Komponenten wurde auf einfache Handhabung geachtet. Aus diesem Grund wurden bedrahtete Element und die MCP23017 im DIP28-Gehäuse verwendet.

Auch die Software mit der Schaltplan und Layout entworfen wurden sollte für jeden frei verfügbar sein um eine möglichst große Zielgruppe ansprechen zu können. Aus diesem Grund wurde das Schaltungsdesign komplett mit EAGLE erzeugt.

3 Schaltplan

Schaltpläne stellen eine schematische Darstellung der Bauelemente und deren Verbindungen untereinander dar. Die folgenden Abschnitte erörtern die Entwicklungsergebnisse der mit EAGLE erstellten Schaltpläne. Es werden die der Entwicklung zugrunde liegenden Grundsätze, so wie Besonderheiten der erstellten Lösungen und Bauteile erörtert.

3.1 Port-Adapter

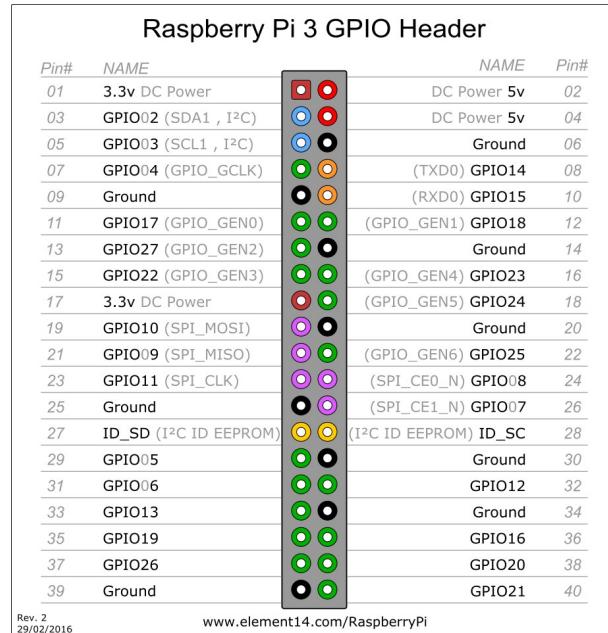


Abbildung 3.1: Raspberry Pi 3 GPIO Header

Gemäß der Informationen von [I5] sind die GPIO Header des Raspberry Pi 3 wie auf Abbildung 3.1 spezifiziert zu finden. Diese GPIO Header werden vom Baustein X1, einer Female-Stiftleiste, abgenommen und so auf dem Port-Adapter zur Verfügung gestellt.

X6 und X7 sind mit dem Step Down Konverter verbunden. Diese müssen gemeinsam an der korrekten Position auf dem Adapter-Port angeordnet werden. Dies erspart das Anlegen des Step Down Konverters in der Bauteil Datenbank von EAGLE. Der Step Down Konverter kann entweder direkt mit diesen Anschlüssen verbunden werden oder mittels Stiftleisten als aufgestecktes Modul ausgelegt werden. Es ist zu beachten, dass der Step Down Konverter noch wie in Abbildung 2.3 erörtert durch schneiden der Verbindung und setzen einer Lötbrücke auf eine 3.3 Volt Festspannung konfiguriert wird.

Um die Game-Pads mit den Raspberry Pi 3 GPIO Headern und der 3v3 Versorgung des Step Down Konverters zu verbinden dienen die 2x5 IDC-Stiftleisten X2, X3, X4 und X5. Diese sind unbedingt mit Verpolungssicherheit auszulegen, da sonst eine Beschädigung der Hardware droht. Die Belegung der einzelnen Pins wurde auf Abbildung 2.4 bereits erörtert. Sie ist maßgeblich durch den verfügbaren Platz auf dem Game-Pad und der daher stammenden Belegung der Pins am Game-Pad bestimmt.

Die Hohlbuchse J1 dient zur Stromversorgung des Systems. Durch die Verdrahtung müssen diesem Anschluss 5 Volt bei ca. 4 Ampere zugeführt werden. Auch wenn der Step Down Konverter bis zu 30 Volt tolerant ist, wird der Raspberry Pi 3 bereits bei kleineren Überschreitungen Schaden nehmen.

Das Bauteil JP1 ist ein Jumper welcher platinenseitig durchkontakteert ist. Durch schneiden der Verbindung kann der Raspberry Pi separat von der Hohlbuchse mit Strom versorgt werden. Da dies im Normalfall unnötig sein sollte, ist die Kontaktierung bereits auf der Platine vorgesehen.

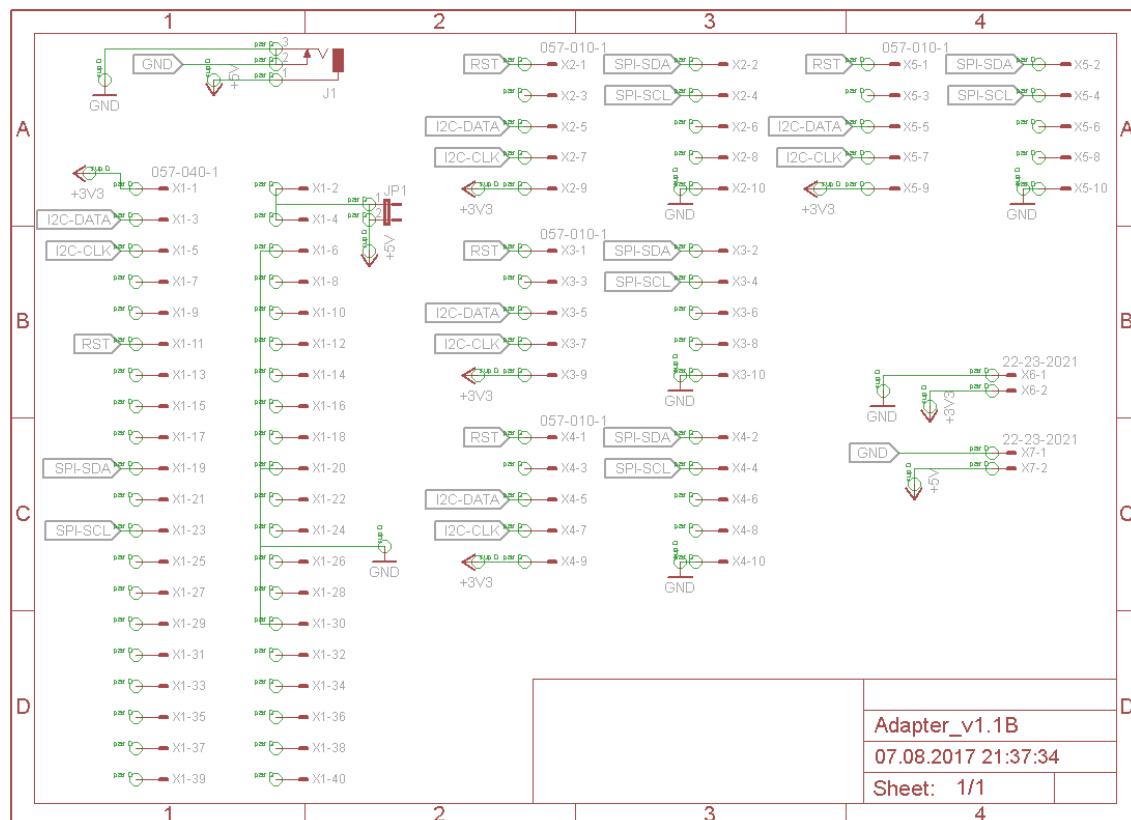


Abbildung 3.2: Schaltplan zum Port-Adapter

Die obere Abbildung 3.2 zeigt den Schaltplan des benötigten Systems. Diese Informationen können in der Datei „eagle\Port\Adapter_v1.1B.sch“ der beigefügten DVD eingesehen werden.

3.2 Game-Pad

Bevor das Game-Pad gelayoutet wurde, mussten zunächst gewisse Vorarbeiten geleistet werden. So wurden das TFT-Display und die Taster als Bausteine in der Bausteindatenbank von EAGLE ergänzt. Besonders zum Abschätzen des Platzbedarfs und zum Vorsehen der Befestigungen im späteren Layout wird ein korrektes Abbild in der Bausteindatenbank benötigt.

Wie sich am ersten Prototypen herausstellte ist die Belegung der Pins am Adressencoder RENC1 in der Bausteindatenbank fehlerhaft, dies wurde zunächst auf einfache Weise korrigiert und ist in der aktuellen Version des Schaltplans beachtet und mittels Anpassung der Belegung korrigiert.

Der Widerstand R1 fungiert als Vorwiderstand für die Licht emittierende Diode LED1. Mittels LED1 wird das Überschreiten des Adressbereichs signalisiert.

Die Widerstände R2, R3 und R4 dienen Als Pull-Down-Widerstände für die Adressierung des Port-Expanders MCP23017. Das High-Signal für die Adressierung stammt vom bereits erwähnten RENC1.

Das WIRE Bauteil ist der Anschlusspunkt für das Flachkabel des Port-Adapters. Da Teile des Layouts wie zum Beispiel die Position diverser Bauteile bereits vor der Entwicklung des Schaltplans feststanden konnten diese Informationen aktiv in die Planung der Busbelegung einfließen. Ein grobes Layout entstand so bereits auf Papier vor dem Schaltplan. Dieses wurde jedoch während des Layout-Prozesses mehrmals angepasst um den Ansprüchen zu genügen und eine optimale Raumnutzung zu gewährleisten.

Die Bausteine U\$2 bis U\$13 sind die Taster des Game-Pad. Sie können vom MCP23017, welcher als IC1 bezeichnet wird, als Eingang gelesen werden. Die Signale BL, CS, und DC dienen zur Steuerung des TFT-Displays.

Das TFT-Displaymodul befindet sich in der oberen Mitte des Schaltplans. Es kann sowohl über SPI als auch über I²C angesprochen werden. Um keine spürbare Überlast auf dem I²C-Bus hervor zu rufen wird die Displaykommunikation über den SPI-Bus geleitet. Diese Aufteilung in zwei Bussysteme garantiert, dass die Game-Pads auch dann noch sicher gelesen werden können, wenn an die Displays viele Daten übertragen werden.

Die folgende Abbildung 3.3 zeigt den Schaltplan des benötigten Systems. Diese Informationen können in der Datei „eagle\Game-Pad\controller_v1.1B.sch“ der beigefügten DVD eingesehen werden.

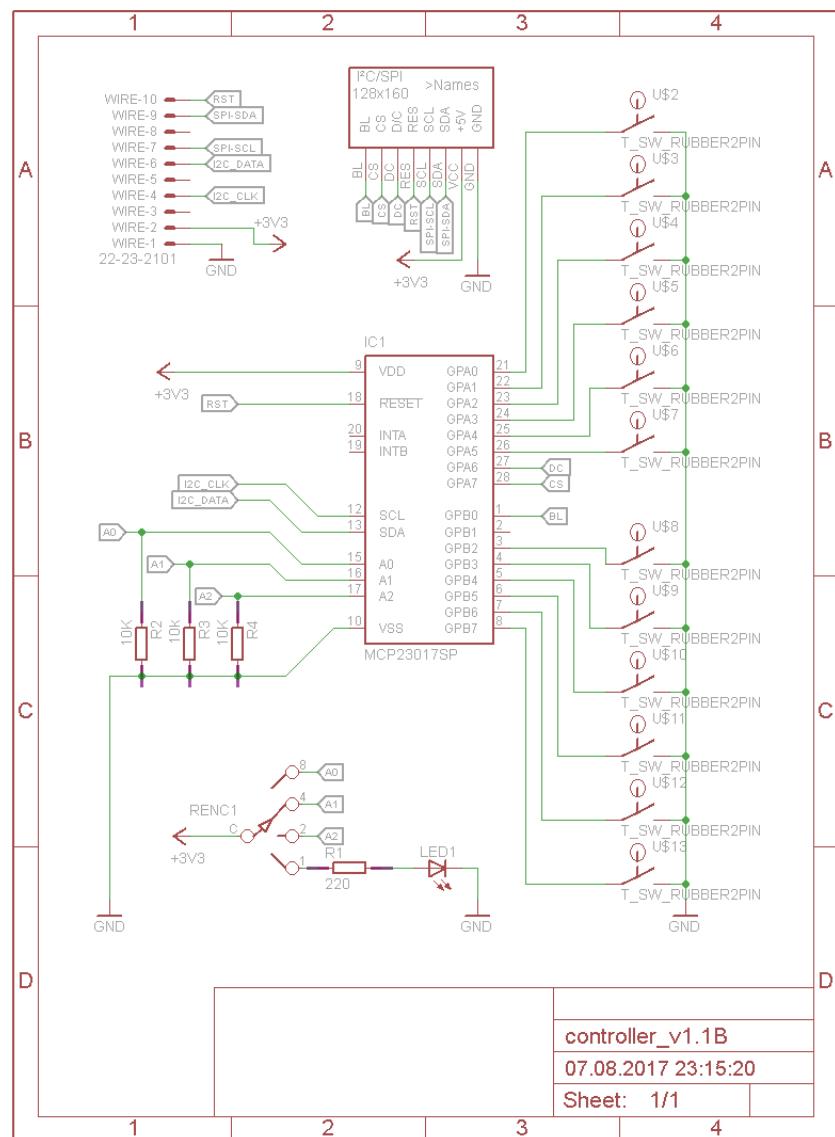


Abbildung 3.3: Schaltplan zum Game-Pad

4 Layout

Die folgenden Abschnitte erörtern die Entwicklungsergebnisse der in EAGLE erstellten Layouts. Die in diesem Schritt erstellten Daten basieren auf den im vorherigen Schritt beschriebenen Schaltplänen.

4.1 Port-Adapter

Der Port-Adapter stellt den Anschlusspunkt der Game-Pads da. Diese werden durch die hier umgesetzte Verdrahtung mit den Anschlüssen des Raspberry Pi 3 verbunden. Diese Anschlüsse der Game-Pads sind die aus Kapitel 3.1 bekannte Parallelverdrahtung zu den vom Raspberry Pi 3 herführenden Pins. Belegt wurden sie mit der in Kapitel 2.3 vorgestellten Pinbelegung. Abbildung 2.4 und Abbildung 2.5 zeigen die Pinbelegung dieser Anschlüsse. Zu beachten ist, dass die in der Abbildung erkennbare Lücke am Gehäuse der Buchse einen Schutz vor Verpolung bietet. Dieser muss bei der späteren Bestückung korrekt auf die Platine gelötet werden um die korrekte Funktionsweise zu gewährleisten.

Der Parallel-Bus wurde in Form einer Kammstruktur ausgelegt. Diese Strukturierung ermögliche eine einfache Erweiterung der bestehenden Platine. Die derzeitige Platine des Port-Adapters verfügt über Anschlüsse für vier Game-Pads und überragt den Raspberry Pi dabei um nur wenige Zentimeter. Da die vom Raspberry Pi besonders hochstehenden Anschlüsse für Ethernet und USB-Buchsen bis zur Platine ragen, wurden die Maße so gewählt, dass die Platine des Port-Adapters vor diesen Anschläßen endet.

Eine Erweiterung auf acht Anschlüsse ist möglich. Mehr als acht Game-Pads pro Raspberry Pi sind jedoch nicht möglich. Die Adressierung der Port-Expander verhindert die Erweiterung auf mehr als acht Game-Pads. Auch der später beschriebene Treiber wird daher nur acht Game-Pads unterstützen.

Die extra breiten Leitungen ermöglichen es diese Platine auch im Tonertransferdruckverfahren zu reproduzieren. Die Anzahl der Durchkontakteierungen konnte an dieser Stelle aufgrund der verwendeten Kammstruktur leider nicht minimiert werden.

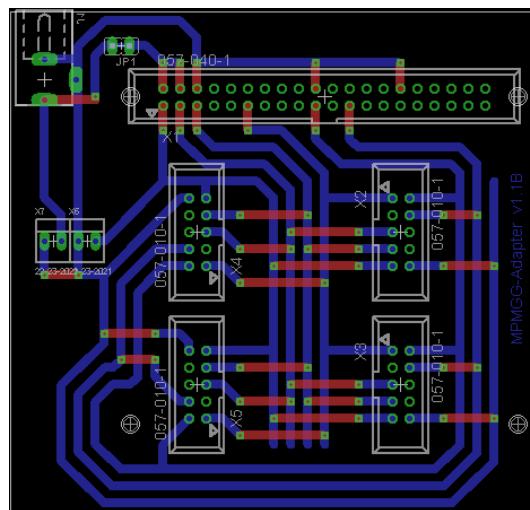


Abbildung 4.1: Layout zum Port-Adapter

Abbildung 4.1 zeigt das mit EAGLE erstellte Platinenlayout. Die blauen Leitungen stellen die Verbindungen auf der unteren Seite der Platine dar. Die roten Leitungen stellen die Verbindungen auf der oberen Seite der Platine dar. Anschlusspads und Durchkontaktierung werden grün gezeichnet. Die geplanten Bauteile, so wie Bohrungen werden grau gezeichnet. Einen genauen Überblick bietet die auf der beigelegten DVD unter „eagle\Port“ zu findenden Datei „Adapter_v1.1B.brd“. EAGLE ermöglicht es die verschiedenen Darstellungsebenen einzeln zu aktivieren und zu deaktivieren.

4.2 Controller-Pad

Die Position der Bauteile ist entscheidend für die spätere Haptik. Das Game-Pad sollte möglichst kompakt, allerdings nicht zu klein für eine anständige Handhabung sein. Zudem müssen alle Komponenten einen Platz finden und die Taster erreichbar sein. Herausragende Komponenten bieten die Gefahr beim Betrieb beschädigt oder zerstört zu werden.

Die Komponenten wurden derart gewählt, dass sie auch für Laien und ohne spezielles Werkzeug gut zu verarbeiten sind. So wurde der MCP23017 im DIP28-Gehäuse eingesetzt. Die Widerstände sind ebenso bedrahtet. Dies hat als Nebeneffekt, dass zusätzliche Durchkontakteungen gespart werden konnten.

Die beiden Taster am oberen Rand des Game-Pad sind zwar als auf der Oberfläche befindlich gezeichnet, jedoch bietet die Wahl von Tastern welche ebenfalls bedrahtet sind auch die Möglichkeit sie wie in der späteren Abbildung 4.5 gezeigt als stirnseitige Variante verbauen zu können.

Die extra breiten Leitungen ermöglichen es diese Platine auch im Tonertransferdruckverfahren zu reproduzieren. Die Anzahl der Durchkontakteungen wurde nach Möglichkeit minimal gehalten, da die feinen Bohrungen besonders kleine Bohrer und hiermit spezielles Werkzeug erfordern.

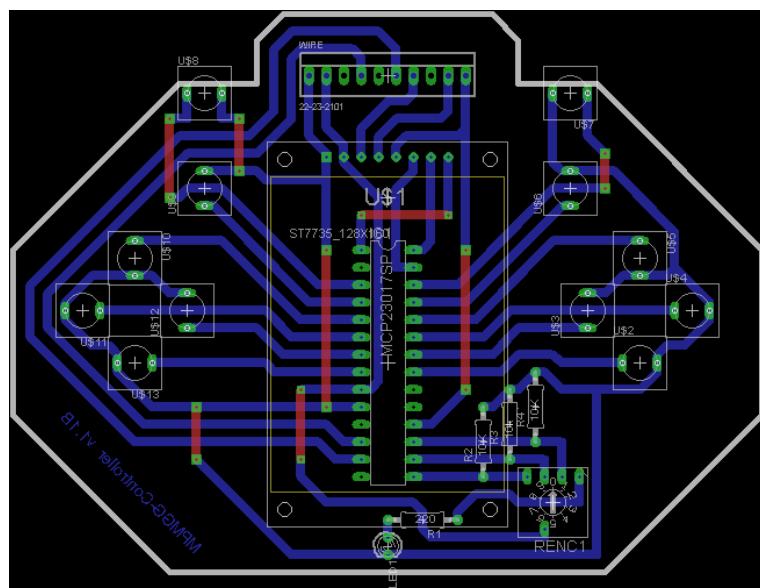


Abbildung 4.2: Layout zum Controller-Pad

Abbildung 4.2 zeigt das mit EAGLE erstellte Platinenlayout. Die blauen Leitungen

stellen die Verbindungen auf der unteren Seite der Platine da. Die roten Leitungen stellen die Verbindungen auf der oberen Seite der Platine da. Anschlusspads und Durchkontaktierung werden grün gezeichnet. Die geplanten Bauteile, so wie Bohrungen werden grau gezeichnet. Einen genauen Überblick bietet die auf der beigelegten DVD unter „eagle\Game-Pad“ zu findenden Datei „controller_v1.1B.brd“. EAGLE ermöglicht es die verschieben Darstellungsebenen einzeln zu aktivieren und zu deaktivieren. Das erzeugte Layout des Controller-Pad reicht in den Breite an die maximale Grenze des in der freien Version von EAGLE zur Verfügung stehenden Platinenbereichs.

4.3 Herstellung

Die folgenden Abschnitte beschreiben den Herstellungsprozess und das daraus resultierende Ergebnis, die Prototypen.

4.3.1 Fräsen

Auch wenn eine Reproduktion mittels Tonertransferdruck möglich ist, wurden die ersten Platinen der für diese Projektarbeit verwendeten Prototypen mittels Fräsen hergestellt. Das Produkt der gefrästen Platinen ist präziser und weniger fehleranfällig als die mittels Tonertransferdruck hergestellte Referenz. Beim Fräsen können zweiseitige Platinen hergestellt werden. Dieser Vorgang benötigt zwei Bohrungen als Referenzpunkte im zweiseitig kupferbeschichtetem Basismaterial, diese werden im Fräsvorlauf automatisch gesetzt und nach der Drehung für die Bearbeitung der Rückseite für eine exakte Positionierung mittels einer Kamera gesucht. Diese Herstellungsvariante bearbeitet nicht nur die Oberfläche des Basismaterials, sondern setzt auch im Layout-Tool definierte Bohrungen und schneidet die äußere Linie des Layouts. Die technischen Anlagen für die Herstellung der Prototypen wurde vom Fachgebiet „Rechnerarchitektur und Systemprogrammierung“ der Universität Kassel zur Verfügung gestellt. Die späteren Prototypen sind mittels industriellen Fabrikationsprozessen hergestellt.

4.3.2 Handbestückung

Die Bauteile der vorbereiteten Platinen wurden von Hand aufgebracht. Bei diesem Arbeitsschritt werden nacheinander von den kleinsten zu den größten Bauteilen alle Komponenten mit der Platine verlötet.

Hierfür werden zunächst die Durchkontakteierungen hergestellt. Es werden kleine unisolierte Drähte durch die hierfür vorgesehenen Bohrungen geschoben und auf beiden Seiten verlötet.

Die nächst kleineren Bauteile sind Stift-, Buchsenleisten und Präzisionsstifitleisten. Diese werden durch die hierfür vorgesehenen Bohrungen gesteckt und mit der Platine verlötet. Bei Buchsen mit Verpolschutz muss auf die korrekte Position dieses Schutzes geachtet werden um korrekt belegte Pins zu erhalten. Bei den in diesem Arbeitsschritt aufgebrachten Bauteilen ist auf eine möglichst orthogonale Ausrichtung zur Platine zu achten um den korrekten Sitz der auf ihnen gesteckten Komponenten zu gewährleisten.

Die benötigten Widerstände, Taster und die LED können nun zurecht gebogen, eingesteckt und verlötet werden. Die letzten zu verlötzenden Bauteile sind der Adressencoder des Game-Pad und die Hohlbuchse des Port-Adapters.

Im letzten Schritt können die zusätzlichen Module eingesteckt werden. Zu ihnen gehören am Game-Pad der Portexpander MCP23017 und das 1.8 Zoll TFT-Modul, so wie am Port-Adapter der Raspberry Pi 3 und der vorbereitete Step Down Konverter.

4.3.3 Prototypen

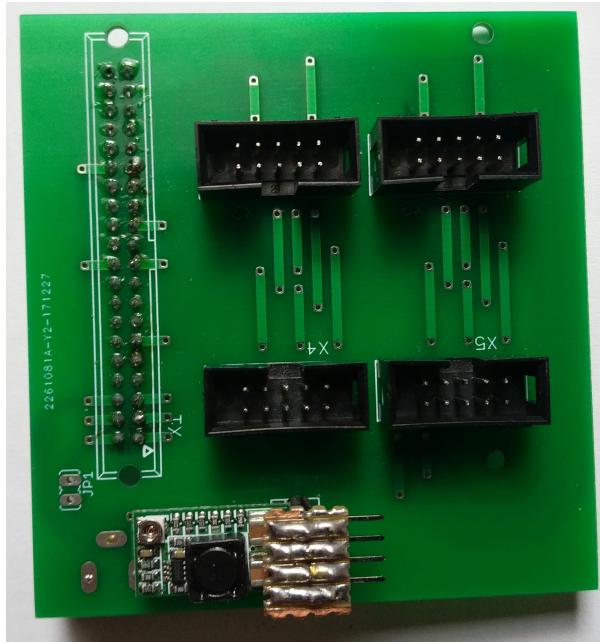


Abbildung 4.3: Prototyp des Port-Adapters

Auf Abbildung 4.3 ist der fertig bestückte Port-Adapter zu sehen. Dies ist ein Prototyp der zweiten Generation. Die Platine wurde in diesem Fall nicht gefräst sondern professionell geätzt. Diese kann direkt auf den Raspberry Pi 3 gesteckt werden. Für den Betrieb mit dem Step Down Konverter müssen die 3v3 GPIO-Pins des Raspberry Pi 3 vom Verbindungsnetz des Bus getrennt werden und der EN-Pin des Step Down Konverters von der GND Netz getrennt werden. Die untere Abbildung 4.4 zeigt die Unterseite des Port-Adapters, die beiden Pfeile zeigen die zu kappenden Verbindungen. Der rote Pfeil zeigt die Trennung des 3v3 Pin zu dem von den Controllern genutzten gemeinsamen Bus. Der gelbe Pfeil Zeigt die Trennung zwischen dem EN-Pin des Step Down Konverters und vom Raspberry Pi wie auch den Controllern genutzten GND Netz.

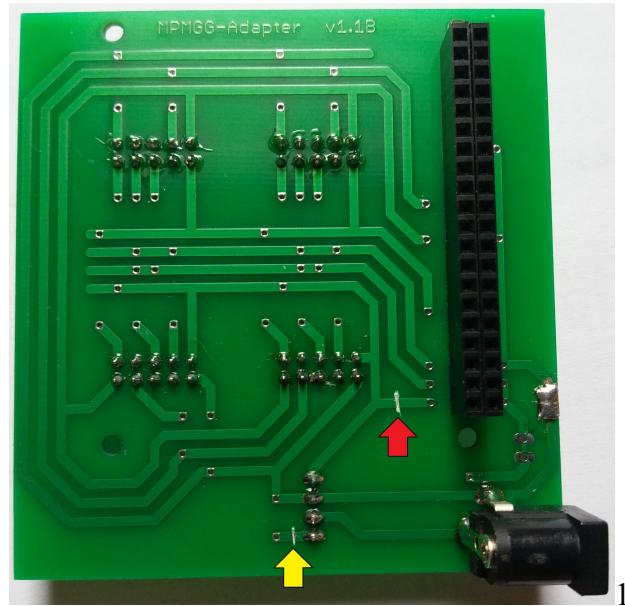


Abbildung 4.4: Port-Adapter mit den nötigen Schnitten

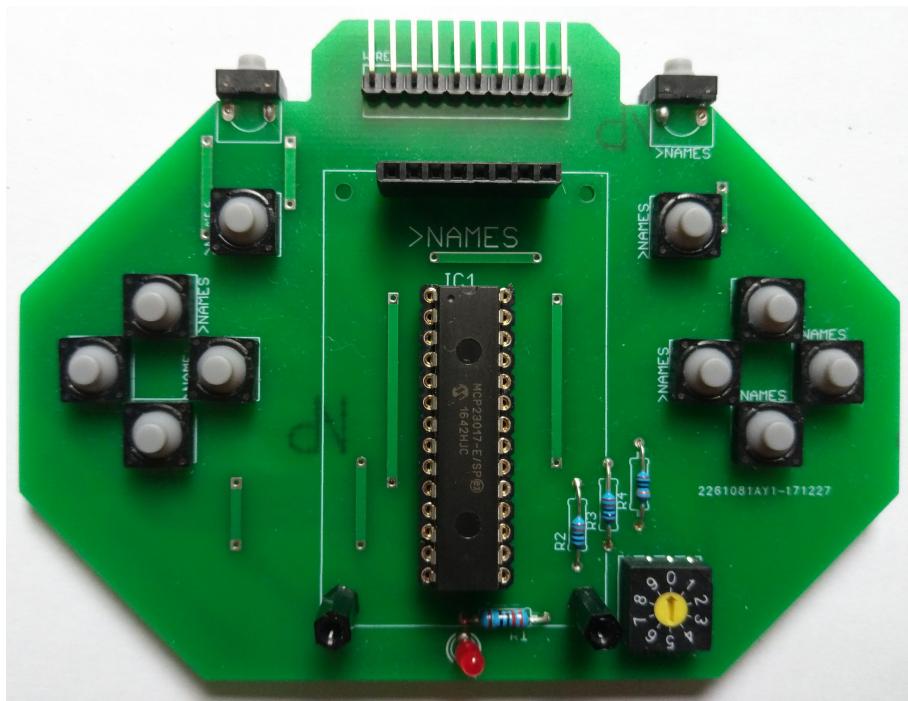


Abbildung 4.5: Prototyp des Game-Pad ohne Display

Abbildung 4.5 zeigt ein fertig bestücktes Game-Pad. Das 1.8 Zoll TFT-Modul wurde entfernt um die Position und Ausrichtung des Port Expander MCP23017 zu zeigen. Die oberen beiden Taster wurden als mit den Zeigefingern zu bedienende Top-Tasten orthogonal zur Platinen verlötet.

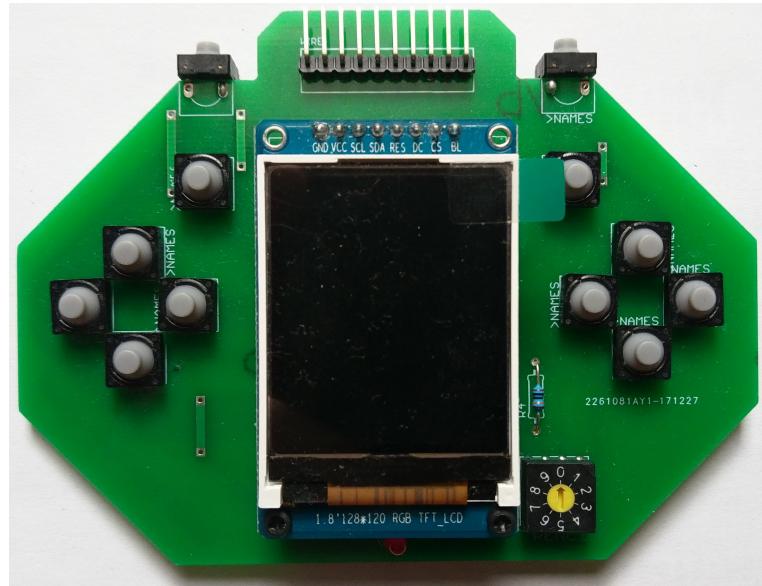


Abbildung 4.6: Prototyp des Game-Pad (montiert)

Die obere Abbildung 4.6 zeigt abschließend das Game-Pad mit 1.8 Zoll TFT-Modul und eingestecktem Anschlusskabel zum Port-Adapter. Für eine bessere Handhabung wurde die Platine auf eine dünne Holzunterlage montiert. Dieser Prtotypenaufbau lässt die finalen Abmessungen eines Game-Pads im Gehäuse erahnen.

Die untere Abbildung 4.7: Prototyp des Game-Pad, Rückseite zeigt die Rückseite des Game-Pads. Die beiden Schrauben am unteren Rand der Platine halten das Display mittels Distanzhülsen. Auf die obere Befestigung des Displays wird verzichtet, da das Display an dieser Stelle durch den Steckverbinder der (Stift-/Buchsenleiste) gehalten wird.

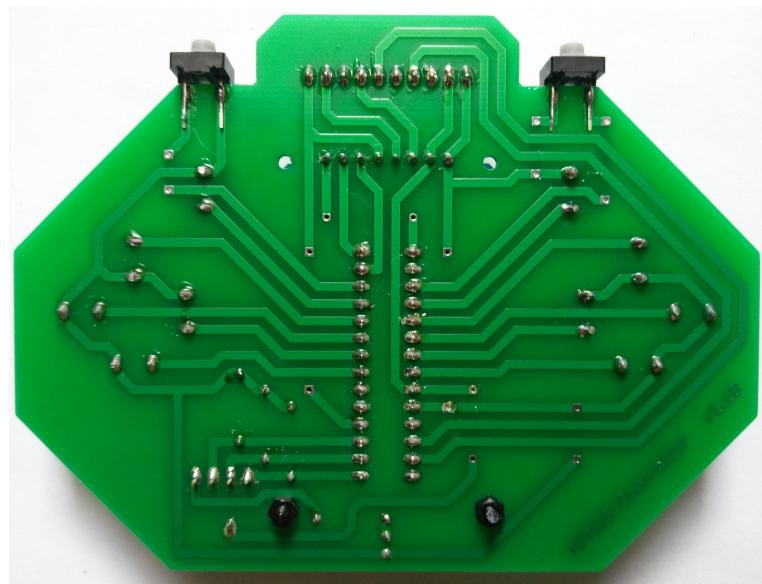


Abbildung 4.7: Prototyp des Game-Pad, Rückseite

5 Erster Test

Um Produktionsfehler der vorbereiteten Hardware aus zu schließen wurden die Komponenten einzeln in Betrieb genommen und getestet. Die hierfür notwendigen Schritte werden in den folgenden Kapiteln beschrieben.

5.1 Vorbereiten des Raspberry Pi und Test der Betriebsbereitschaft

Der Raspberry Pi 3 benötigt zum Betrieb ein Kernel-Image auf einer MikroSD-Karte. Für dieses Projekt wurde ein fertig vorbereitetes Image von Ubuntu MATE 16.04.2 für den Raspberry Pi 2 und 3 verwendet. Dieses Image kann von [I6] bezogen werden.

Nach dem Download kann das Image mittels Win32DiskImager, welcher von [I7] bezogen werden kann, auf eine MikroSD-Karte geschrieben werden.

Nachdem der RPI3 mit einem TV-Gerät verbunden, die MikroSD-Karte eingesteckt und eine Stromversorgung hergestellt wurde begann wie erwartet der Bootvorgang.

Nach Abschluss des Bootvorgangs kann die Konfiguration des RPI3 beginnen. Hierfür müssen zunächst die SPI- und I²C-Schnittstelle aktiviert werden. Hierfür wurde Methode 2 aus [I8] verwendet.

Ebenso wurde das Paket *i2ctools* installiert um den I²C-Bus genauer untersuchen und eine erste Kommunikation aufzubauen zu können. Um das Packet *i2ctools* zu installieren wird eine Konsole gestartet und der folgende Befehl eingegeben:

```
sudo apt-get -y install i2ctools
```

5.2 Test des MCP23017 mit i2ctools

Der an dem Game-Pad verwendete Portexpander kann mittels dem Paket *i2ctools* auch ohne Treiber getestet werden. Hierfür muss das Game-Pad mir dem Port-Adapter und dieser mit dem Raspberry Pi verbunden sein und eine beliebige Adresse am Addressencoder eingestellt sein.

Durch den Aufruf des zum Paket *i2ctools* gehörenden Kommandos *i2cdetect* kann zunächst der Bus auf seine angeschlossenen Komponenten untersucht werden. Hierfür muss der folgende Befehl in die Konsole eingegeben werden:

```
i2cdetect -y 1
```

Dieser Aufruf liefert eine Tabelle mit Informationen über die am Bus befindlichen Geräte welche eine Antwort auf die ihnen zugeteilte Adresse liefern. Ein Beispiel für einen solchen Aufruf ist die Ausgabe:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00:			--	--	--	--	--	--	--	--	--	--	--	--	--	--
10:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
20:	20	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
30:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
40:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
50:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
60:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
70:	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Sie bedeutet, dass an Adresse 0x20 ein angeschlossenes Gerät die Nachricht bestätigt. Die MCP23017 Schaltung besitzt eine Grundadresse von 0x20, auf welche die am Adressencoder eingestellte Adresse addiert wird. Der Adressencoder steht somit auf der Einstellung „0“.

Die 16 GPIO Pins des Port Expanders sind in 2 Register aufgeteilt, GPIOA auf Registeradresse 0x12 und GPIOB auf Registeradresse 0x13. Diese können mittels i2cget abgefragt werden. Hierfür muss der folgende Befehl in die Konsole eingegeben werden:

```
i2cget -y 1 0x20 0x13
```

Der obige Befehl fragt nun die Registeradresse 0x13 am Gerät 0x20 an und gibt diese aus. Ein Beispiel für einen solchen Aufruf ist die Ausgabe:

```
0x05
```

Die obige Antwort bedeutet, dass am Gerät 0x20 an der GPIO-Bank B die Pins 1 und 4 an ein HIGH Signal angelegt sind.

Auch das Schreiben an eine Registeradresse des Port Expanders ist mittels i2ctools möglich. Hierfür muss der folgende Befehl in die Konsole eingegeben werden:

```
i2cset -y 1 0x20 0x14 0x01
```

Der obige Befehl schreibt nun am Gerät 0x20 an die Registeradresse 0x14, was dem Latch für GPIO-Bank A entspricht, ein HIGH Signal an den ersten Pin und ein LOW Signal an alle anderen. Voraussetzung hierfür wäre ,dass die GPIO-Pins von Port A des Port Expanders als Ausgänge definiert sind.

Für den ersten Test wurde eine fliegende Verdrahtung gewählt, bei der die nicht vom I²C-Bus angesteuerten Pins auf korrekte Signale gelegt wurden. Bei diesem Test konnte kein positives Ergebnis erzielt werden. Dies hatte den Hintergrund, dass der Portexpander über 5 Volt versorgt wurde. Dies führte allerdings dazu, dass die Schutzdiode des Raspberry Pi das Signal auf HIGH zogen und daher nur falsche Werte vom Bus gelesen werden konnten. Dieses Experiment hatte zur Folge, dass wie oben

beschrieben in der aktuellen Version der Pinbelegung des Port-Adapters die Versorgungsspannung der Game-Pads auf 3v3 geändert wurde. Der erneute Test ergab, dass der auf diese Weise angeschlossene MCP23017 korrekt funktioniert.

Als nächster Schritt wurde eine korrigierte Version des Port-Adapters getestet. Hierfür muss zusätzlich der Reset-Pin des Busses auf HIGH gesetzt werden. Dies wurde wie in [I9] mit Variante 1 beschrieben getestet. Dieser Test ergab, dass der auf diese Weise angeschlossene MCP23017 korrekt funktioniert und auch über den Bus zurück gesetzt werden kann.

5.3 Testen des Displays

Ein Test des Display-Moduls wurde auf Grund der Komplexität des ST7735-Protokolls am RPI3 nicht durchgeführt. Die Betriebsfähigkeit wurde jedoch mittels einem der Beispiele aus der Arduino-Bibliothek verifiziert. Der eigentliche Test des Displays wurde manuell vorgenommen, nachdem der Treiber in Betrieb genommen und mit der erforderlichen Konfiguration versorgt wurde.

6 Treiberentwicklung

Für die Verwaltung der verwendeten Hardware werden spezielle Softwarebestandteile benötigt. Diese werden Kernelmodule genannt. Diese stellen eine Verbindung zwischen den Funktionen des Kernels und der eigentlichen Hardware dar. Sie kapseln nicht nur die Schritte um entsprechende Hardware zu verwenden, sondern informieren das Betriebssystem auch über die möglichen Zugriffsmethoden, reserviert Speicher und melden das Gerät an der Schnittstelle an. Die folgenden Abschnitte beschreiben die Entwicklung der benötigten Treiber und die ihnen zugrunde liegenden Strukturen.

6.1 Kernel-Header installieren

Eine Kernelmodul muss speziell zu einer Kernel-Version kompiliert werden. Hierfür müssen die benötigten Schnittstellen als Ressourcen vorliegen. Die Ressourcen eines Kernels lassen sich mittels dem folgenden Befehl herunterladen:

```
rpi-source
```

Sollte das rpi-source-Kommando nicht in der Konsole installiert sein, so kann es mittels der Anleitung von [I10] nachträglich den Kommandos hinzugefügt werden.

Diese Ressourcen werden von der GNU Compiler Collection (GCC) als System-Header erkannt. Der Präprozessor liest sie an den vorgesehenen Stellen im Quellcode durch eine **#include**-Anweisung ein und erweitert das Kompilat um die entsprechende Datei. [I11] Es werden folgende Systemheader verwendet:

- Für das Kernelmodul „**mpmgg-port**“:
 - *linux/spinlock.h* für die Sicherung bei wechselseitigen Zugriff.
 - *linux/spi/spi.h* für die Zugriffsfunktionen auf den SPI-Bus des RPI3.
 - *linux/platform_device.h* für die Bereitstellung der Platformgerätestrukturen.
 - *linux/module.h* für die Bereitstellung der grundlegenden Kernelmodulstrukturen.
 - *linux/i2c.h* für die Zugriffsfunktionen auf den I²C-Bus des RPI3.
 - *linux/gpio.h* für den Zugriff auf die GPIO's des RPI3.
 - *linux/delay.h* für das bereitstellen von Wartefunktionen.
 - *linux/slab.h* für das Reservieren des Kernelspeichers.
- Für das Kernelmodul „**mpmgg-pad**“:
 - *linux/slab.h* für das Reservieren des Kernelspeichers.
 - *linux/module.h* für die Bereitstellung der grundlegenden Kernelmodulstrukturen.
 - *linux/kernel.h* für notwendige Definitionen im des Kernels.
 - *linux/input.h* für das Verwenden der standardisierten Systeminputs.

- *linux/i2c.h* für die Zugriffsfunktionen auf den I²C-Bus des RPI3.
- *linux/timer.h* für zeitgesteuerte Systemfunktionen.
- *linux/err.h* für die standardisierten Fehlercodes des Kernels.
- *linux/kthread.h* für das erzeugen eines Kernelthreads.
- *linux/freezer.h* für das Benutzen von Systemsteuerungsbefehlen. des *Schedulers*.
- *linux/sched.h* für das Benutzen von Systemsteuerungsbefehlen des *Schedulers*.
- *linux/device.h* für das Anmelden einer Gerätestruktur am System.
- *linux/spi/spi.h* für das bereitstellen von Wartefunktionen.
- *linux/platform_device.h* für die Bereitstellung der Platformgerätestrukturen.
- *linux/bitops.h* für Bitoperationen welche für die Datensicherung verwendet werden.
- *linux/delay.h* für das bereitstellen von Wartefunktionen.
- Für das Kernelmodul „**mpmgg-st7735**“:
 - *linux/spinlock.h* für die Sicherung bei wechselseitigen Zugriff.
 - *linux/platform_device.h* für die Bereitstellung der Platformgerätestrukturen.
 - *linux/module.h* für die Bereitstellung der grundlegenden Kernelmodulstrukturen.
 - *linux/export.h* für das exportieren von Funktionen das Kernelmoduls.
 - *linux/errno.h* für die standardisierten Fehlercodes des Kernels.
 - *linux/kernel.h* für notwendige Definitionen im des Kernels.
 - *linux/vmalloc.h* für das Reservieren von Videospeicher im System.
 - *linux/slab.h* für das Reservieren des Kernelspeichers.
 - *linux/fb.h* für die Verwendung von Funktionen und Strukturen welche den Framebufferbetreffen.
 - *linux/spi/spi.h* für die Zugriffsfunktionen auf den SPI-Bus des RPI3.
 - *linux/uaccess.h* für die Zugriffsfunktionen auf den Userspace des Speichers.
 - *linux/platform_device.h* für die Bereitstellung der Platformgerätestrukturen.
 - *linux/delay.h* für das bereitstellen von Wartefunktionen.

6.2 Konzepte

Die Steuerung der Hardware geschieht mittels drei Kernelmodulen. Diese repräsentieren sogleich die existierende Hardwarestruktur. In den folgenden Kapiteln werden die verwendeten Konzepte beschrieben, um einen Überblick des erarbeiteten

Wissens und ein grundlegendes Verständnis der Umsetzung zu geben.

6.2.1 Port-Adapter

Das Kernelmodul mit dem Namen „mpmgg-port“ dient zur Initialisierung, Erzeugung der benötigten Strukturen und der Verteilung dieser an die weiteren Komponenten.

Dieses Kernelmodul besitzt die Kontrolle über die Steuerung des Hardware-Reset und wurde in der späten Entwicklungsphase um die Initialisierung der Hardwarekomponenten erweitert.

Der als GPIO-Pin ausgelegte Hardware-Reset wird vom RPI3 an der Hardware des Port-Adapters parallel an alle Game-Pads ausgeführt. An den Game-Pads wird dieser Pin sowohl an den Reset Pin des TFT-Displays als auch an den Reset Pin des MCP23017 geführt.

Die benötigten gemeinsamen Strukturen der Game-Pads werden hier erzeugt und mittels Konfiguration von Plattform Geräten an die Treibermodule der Game-Pads übergeben. Des weiteren werden die Game-Pads in ausreichender Quantität am Betriebssystem angemeldet. Geräte, welche nicht am System angeschlossen sind, werden versucht am Kernel an zu melden, jedoch wird dies vom „mpmgg-pad“-Treiber der anzulegenden Struktur verweigert.

Die in der späten Entwicklung hinzugefügte Funktionalität der Hardware-Initialisierung wird vor der Erzeugung der Plattform-Geräte vorgenommen. Hierzu werden zunächst die benötigten Steuerregister des MCP23017 blind beschrieben. Es werden also die Konfigurationen aller Geräte über den Bus versendet ohne zu prüfen ob sich die Empfänger wirklich am Bus befinden. Die genaue Registerkonfiguration kann im Kapitel 6.3 (Zugriff auf MCP23017) eingesehen werden.

Die Initialisierung der Displays geschieht synchron. Es werden zunächst alle benötigten Steuerleitungen aller potenziell am I²C-Bus befindlichen MCP23017's geschaltet und im Anschluss die Konfiguration via SPI an die ST7735 übertragen. Da alle Displays die selbe Konfiguration erhalten und sie sich anhand der MCP23017 später separat adressieren lassen ist eine synchrone Initialisierung möglich. Dieser Mechanismus verkürzt die Zeit, welche eingesetzt werden muss um die Displays mit ihrer Konfiguration zu versehen erheblich. Die genaue Konfiguration der Register kann in Kapitel 6.6.3 (Implementierung des Display-Treibers „mpmgg-st7735“) nachgelesen werden.

Die zugehörigen Dateien sind „mpmgg-port.h“ für die benötigten Deklarationen des Port-Adapters, mpmgg-pad-control.h für die Registerdefinitionen des MCP23017, mpmgg-st7735.h für die Initialisierungssequenz des ST7735-Displaymoduls, „mpmgg-port.c“ für die benötigte Implementierung der Definitionen und „mpmgg-xchange.h“ für alle Arten von Definitionen, welche die abgeschlossenen Kernelmodule verbinden.

6.2.2 Game-Pad

Das Kernelmodul mit dem Namen „mpmgg-pad“ dient zur Steuerung des Game-Pad. Es übernimmt die Konfiguration, welche vom „mpmgg-port“ als Plattform Gerät beim Starten übergeben wird.

Zudem verwaltet dieses Modul die benötigten Speicherbereiche und meldet das Gerät als Joystick am Betriebssystem an. Prinzipiell sind dem Treiber das Einspielen aller Eingaben möglich. Es werden jedoch zur Erhaltung der Kompatibilität mit anderen auf dem Markt verfügbaren Game-Pads die Standardeingaben solcher Game-Pads verwendet. Die Eingaben wurden vorher durch Analysieren des „USB SNES Game-Pad/Controller for PC“ von iNNEXT® mittels jstest, welches Teil des in Kapitel 7.2 (Test des Game-Pads) vorgestellten „joystick“ Paket von Linux ist, analysiert.

Das Kernelmodul greift auf den I²C-Bus zu und steuert den MCP23017 Port Expander. Dies ermöglicht den lesenden Zugriff auf die Taster und das Steuern der mit dem Port Expander verbundenen Steuerleitungen des Display-Moduls. Zum Lesen der Tasten des Game-Pads wird zu jedem verfügbaren Game-Pad ein Thread gestartet, welcher mittels polling die Informationen der MCP23017 Port Expander abfragt.

Ein weiterer Schritt beim Starten des Game-Pads ist das Erzeugen der Strukturen für das 1.8 Zoll TFT-Modul. Die für das Display benötigten gemeinsamen Strukturen, welche bereits von mpmgg-port erzeugt wurden (die Semaphoren zum Schutz der Kritischen Abschnitte für den wechselseitigen Zugriff auf den I²C und SPI-Bus), so wie die Adressen der Funktionen für die Manipulation der Signalleitungen werden vom „mpmgg-pad“-Treiber zusammengestellt und mittels Konfiguration von Plattform Geräten an das Treibermodul des Displays übergeben. Die zugehörigen Dateien sind „mpmgg-pad.h“ für die benötigten Definitionen, „mpmgg-pad-control.h“ für die benötigten Register Definitionen und konfiguration, „mpmgg-pad.c“ für die benötigte Implementierung der Definitionen und „mpmgg-xchange.h“ für alle Arten von Definitionen welche die abgeschlossenen Kernelmodule verbinden.

6.2.3 TFT-Display

Das Kernelmodul mit dem Namen „mpmgg-display“ dient zur Steuerung des auf dem Game-Pad angebrachten TFT-Displays. Es übernimmt die Konfiguration welche vom „mpmgg-pad“ als Plattform Gerät beim Starten übergeben wird. Zudem verwaltet dieses Modul die benötigten Speicherbereiche und meldet das Gerät als Framebuffer am Betriebssystem an.

Framebuffer beschreiben grafische Ausgabegeräte, welche pixelorientierte Grafiken erhalten, diese auf ein für die Hardware der Ausgabe passenden Format übersetzen und diese an das Ausgabegerät übertragen.

Die Aufgabe des Kernelmoduls ist die Bereitstellung der benötigten Zugriffe der vom Betriebssystem stammenden Funktionen für die Steuerung der Hardware und diesbezüglich auch die Sicherung beim wechselseitigem Zugriff auf den gemeinsam genutzten SPI-Bus.

Die Implementierung ist sehr nah an der für diese Zwecke eruierten „fb-st7735r“ von

[I12]. Für die hier verwendete Implementierung wurde die Struktur an vielen Stellen vereinfacht und an das zuvor definierte Milieu angepasst.

Die zugehörigen Dateien sind „mpmgg-st7735.h“ und „mpmgg-st7735-control.h“ für die Definitionen der benötigten Befehle und Adressen; „mpmgg-st7735-main.c“ für die implementierung der Kernfunktionalitäten, „mpmgg-st7735-fops.c“ für die benötigten Systemzugriffsfunktionen, „mpmgg-st7735-sysfs.c“ für die benötigten Umrechnungsfunktionen und „mpmgg-st7735-support.c“ für die benötigten Buszugriffsfunktionen, so wie „mpmgg-xchange.h“ für alle Arten von Definitionen welche die abgeschlossenen Kernelmodule verbinden.

6.3 Zugriff auf MCP23017

Der MCP23017 Port Expander kann via I²C oder SPI-Bus angesteuert werden. Im Weiteren wird jedoch nur die Ansteuerung mittels I²C-Bus verwendet. Jeder Port Expander wird über eine extern angelegte 3 Bit-Adresse angesprochen. Dies ermöglicht es am I²C-Bus bis zu 8 unterschiedlich adressierte MCP23017 Schaltungen zu erreichen. Es sind die Taktgeschwindigkeiten 100 kHz, 200 kHz, 400 kHz und 1.7 MHz auf dem Bus möglich. [I26]

Wie im Datenblatt auf Abbildung 3-3 zu sehen, muss dem Client zum Lesen von Informationen neben dem für den Bus definierten Startbit, dem Operationscode, dem Lese/Schreibe-Flag und der Geräteadresse noch die ein Byte lange Registeradresse vor dem Stopbit übersendet werden. Das nun auf dem Bus vom Master lesbare Byte ist der Registerinhalt des angefragten Registers.

Wie im Datenblatt auf Abbildung 3-3 zu sehen, muss dem Client zum Schreiben von Informationen in ein Register neben dem für den Bus definierten Startbit, dem Operationscode, dem Lese/Schreibe-Flag und der Geräteadresse noch die ein Byte lange Registeradresse, so wie die zu schreibende Bytes vor dem Stopbit übersendet werden.

TABLE 3-4: CONTROL REGISTER SUMMARY (IOCON.BANK = 1)

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	01	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	02	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	03	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	04	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	05	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	06	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFA	07	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	0000 0000
INTCAPA	08	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	09	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	0A	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
IODIRB	10	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLB	11	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENB	12	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALB	13	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONB	14	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	15	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUB	16	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFB	17	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	0000 0000
INTCAPB	18	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOB	19	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATB	1A	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

TABLE 3-5: CONTROL REGISTER SUMMARY (IOCON.BANK = 0)

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	0000 0000
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	0000 0000
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

Abbildung 6.1: Auszug aus dem Datenblatt des MCP23017

Die Abbildung 6.1 stammt aus dem Datenblatt [I26] und zeigt die beiden Varianten der Registerbelegung des MCP23017. Wird der MCP23017 mit der Standardkonfiguration betrieben so ist das IOCON mit dem Wert 0x00 versehen. Hieraus ergibt sich, dass die Konfiguration mit dem Flag Bank = 0 arbeitet und daher die zweite Tabelle aus

Abbildung 6.1 für die Registeraufteilung zu verwenden ist. Bei den Versuchen und Fehleranalysen am Ende der Entwicklungsphase wurden die auf Grund einer sehr hohen Bitfehlerrate die Konfiguration mit Bank = 1 getestet. Dies erwies sich als Möglichkeit Übertragungsfehler erfolgreich zu vermindern. Daher wurde dies beibehalten und die erste Registerkonfiguration aus Abbildung 6.1 für die folgende Konfiguration der benötigten Register verwendet. Es ist hierbei zu beachten, dass sich die Registeradressen bei unterschiedlichen Konfigurationen des Bank-Flag stark unterscheiden.

Zur Initialisierung des Port Expanders müssen die Register IODIRA (Registeradresse 0x00 bei Bank = 0, Registeradresse 0x00 bei Bank = 1) und IODIRB (Registeradresse 0x01 bei Bank = 0, Registeradresse 0x10 bei Bank = 1) mit den zugehörigen Schreib- und Leserichtungen beschrieben werden. Ein HIGH-Bit bedeutet hierbei, dass der entsprechende Pin ein Eingang ist. Ein LOW-Bit bedeutet, dass der entsprechende Pin ein Ausgang ist. Aus dem Schaltplan ergibt sich somit der Wert 0x3F für Register IODIRA und 0xFC für Register IODIRB. Alle weiteren Registerwerte können den Initialwert beibehalten. Sie werden daher nicht verändert.

Im Weiteren werden die Register GPIOA (Registeradresse 0x12 bei Bank = 0, Registeradresse 0x09 bei Bank = 1) und GPIOB (Registeradresse 0x13 bei Bank = 0, Registeradresse 0x19 bei Bank = 1), so wie OLATA (Registeradresse 0x14 bei Bank = 0, Registeradresse 0x0A bei Bank = 1) und OLATB (Registeradresse 0x15 bei Bank = 0, Registeradresse 0x1A bei Bank = 1) verwendet. In den Registern GPIOA und GPIOB befinden sich Informationen über die derzeit am zugehörigen Pin angelegten Logiklevel der Bänke A und B. Diese Register sollten nur gelesen werden. Sie liefern die Information welche Taster zum Anfragezeitpunkt gedrückt sind. In den Registern OLATA und OLATB befinden sich Informationen über die derzeit auf den zugehörigen Ausgangs-Pin geschriebenen Logiklevel der Ports A und B. Diese werden verwendet um die Steuerpins des Display zu belegen.

Die Register GPPUA (Registeradresse 0x0C bei Bank = 0, Registeradresse 0x06 bei Bank = 1) und GPPUB (Registeradresse 0x0D bei Bank = 0, Registeradresse 0x16 bei Bank = 1) definieren die Verwendung der im MCP23017 integrierten Pullups. Werden diese Bits auf den logischen Level 1 gesetzt, so werden die Pullups verwendet was die Logiklevel der Eingänge im Leerlauf auf den HIGH Pegel zieht. Diese Konfiguration wird für alle Eingänge mit Tastern des Game-Pads verwendet um einen definierte Pegel bei nicht gedrückten Tastern zu erzeugen.

Die Register IPOLA (Registeradresse 0x02 bei Bank = 0, Registeradresse 0x01 bei Bank = 1) und IPOLB (Registeradresse 0x03 bei Bank = 0, Registeradresse 0x11 bei Bank = 1) definierten die Verwendung oder Nichtverwendung des Signalinverters für die Eingangssignale. Da die internen Pullups des MCP23017 genutzt werden ist das Standardlogiclevel der Eingänge HIGH. Die Taster der Game-Pads verbinden die GPIO des MCP23017 mit dem GND der Game-Pads. Beim Drücken der Taster wird nun aus dem HIGH Signal des GPIO an dem der Taster gedrückt wird ein LOW Signal. Das bedeutet, dass ein nicht gedrückter Taster als logische 1 und ein gedrückter Taster als logische 0 erfasst wird. Durch die Verwendung von IPOLA und IPOLB kann das

Signal nun vor der Übertagung korrigiert werden. Beim Drücken der Taste wird somit das Signal als logische 1 und bei einem nicht gedrückten Taster wird als eine logische 0 über den I²C-Bus übertragen. Diese Auswertung hätte auch auf dem Raspberry Pi stattfinden können, die Funktionalität des MCP23017 wurde jedoch bevorzugt.

Tabelle 6.1: Registerkonfiguration des MCP23017

Reihenfolge:	Registername:	Registeradresse:	Standartwert:	Zielwert:
1	IOCON	0x0A	0x00	0x80
2	IODIRA	0x00	0x00	0x3F
3	GPPUA	0x06	0x00	0x3F
4	IPOLA	0x01	0x00	0x3F
5	OLATA	0x0A	0x00	0x3F
6	IODIRB	0x10	0x00	0xFC
7	GPPUB	0x16	0x00	0xFC
8	IPOLB	0x11	0x00	0xFC
9	OLATB	0x1A	0x00	0xFC

Die obige Tabelle 6.1 zeigt die Konfiguration wie sie der Zielhardware entspricht. Zu beachten ist, dass zunächst IOCON beschrieben wird und dort das Bank-Flag auf den Wert True gesetzt wird. Nach diesem Schreibvorgang verändert sich die Registeraufteilung. Alle weiteren Schreibzugriffe in die Register werden nun auf die neue Registerzuteilung vorgenommen. Die Register werden mittels den zugehörigen Masken auf die Zielhardware konfiguriert. Jede Änderung dieser Masken führt zu Fehlern beim Lesen oder Schreiben der GPIO des zugehörigen Portexpanders. Die Masken ergeben sich aus den angeschlossenen Tastern, welche mit den internen Pull-up Widerständen beschaltet werden müssen, so wie den auf LOW zu schaltenden Steuerleitungen der Display-Module. [I26]

Für den wechselseitigen Ausschluss der unabhängig von einander arbeitenden Kernelmodule der Game-Pads wird ein Mutex verwendet. Alle Busteilnehmer die auf den I²C-Bus schreiben oder von ihm lesen sperren vor ihrer Aktion den Mutex und geben ihn erst nach Vollendung der Aktion wieder frei. Das Konzept von wechselseitigem Ausschluss (in der englischen Literatur „mutual exclusion“ genannt) stellt sicher, dass die Zugriffe auf den I²C-Bus nicht durch Schedules (wechsel zwischen den Threads) beeinträchtigt werden.

Um zu prüfen ob sich ein MCP2317 am Bus befindet kann die Initialisierung verwendet werden. Wird ein Wert in eines der Register geschrieben und kann im Anschluss nicht korrekt ausgelesen werden, so liegt ein Gerätefehler vor und das Gerät kann nicht korrekt verwendet werden. Somit kann die weitere Initialisierung abgebrochen werden, und alle vorangegangenen Initialisierungsschritte, wie zum Beispiel das Reservieren von Speicher, das Anmelden am I²C-Bus oder das Registrieren des Eingabegeräts rückgängig gemacht werden.

Die Umsetzung der soeben genannten Funktionen obliegt dem Kernelmodul „mpmgg-pad“ welches als Plattform Gerät implementiert wurde. Ein Kritikpunkt an dieser Umsetzung ist, dass die Konfiguration und Anmeldung von „mpmgg-port“

vorgenommen wird. Durch kleine Änderungen am Code hätte die Unterstützung des neuen „Device Tree“-Konzeptes eingebracht werden können. Auf diese Weise hätte das Kernelmodul „mpmgg-port“ keine weitere Existenzberechtigung. Wie sich herausstellte dient dieses Kernelmodul nur der Verwaltung der „mpmgg-pad“ Treiber, da neben dem Hardware-Reset aller am Bus befindlichen Geräte keine weitere Funktionalität vorgesehen wurde.

6.4 Gameport Schnittstelle

Der Game-Port stellt die physikalische Verbindung der vom Raspberry Pi 3 gestellten Pins zu den Game-Pads. Die hierfür auf den Raspberry Pi zu steckende Platine wird verwendete um die verwendeten Signal- und Versorgungsleitungen als vollständig parallelen Bus an bis zu acht Klienten, die Game-Pads, zu führen. Da die Reihenfolge der Klienten auf Grund der voll parallelen Auslegung des Busses nicht festgelegt ist können die Klienten beliebig an die Steckplätze verteilt werden. Der Verpolschutz der verwendeten 10-poligen Pfostenstecker verhindert die Möglichkeit durch ein falsches Einsticken die Leitungen zu vertauschen. Eine zu beachtende Belegung ist daher an den Anschlüssen des Gameports nicht zu beachten.

Die einzige Bedingung für die Klienten am Bus ist, dass die Game-Pads mittels des Codierschalters unterschiedlich adressiert sind. Der Codierschalter ist auf der unteren rechten Kante der Vorderseite der Game-Pads zu finden. Für die Adressierung stehen auf Grund des MCP23017 die Adressen 0 bis 7 zur Verfügung. Werden die Schaltpositionen 8 oder 9 mittels dem Codierschalters gewählt, so entspricht dies einem Pufferüberlauf der zur Verfügung stehenden acht Adressen. Dieser wird am Game-Pad durch das Leuchten der LED (LED1) signalisiert. Die Adresszuordnung des MCP23017 am I²C-Bus kann der folgenden Tabelle 6.2: Adresszuordnung am I²C-Bus mittels Codierschalter entnommen werden.

Tabelle 6.2: Adresszuordnung am I²C-Bus mittels Codierschalter

Adresse am Codierschalter	Adressbelegung des MCP23017		LED
	hexadezimal	dezimal	
0	0x20	32	Aus
1	0x21	33	Aus
2	0x22	34	Aus
3	0x23	35	Aus
4	0x24	36	Aus
5	0x25	37	Aus
6	0x26	38	Aus
7	0x27	39	Aus
8	0x20	32	Ein
9	0x21	33	Ein

Die zentrale Rolle des MCP23017 in den Game-Pads ermöglicht das Steuern der Displays via gemeinsamer Nutzung aus I²C- und SPI-Bus. Da die Steuerleitungen jedes ST7735-Displays eines Game-Pads mit dem zugehörigen Portexpander (MCP23017) des jeweiligen Game-Pads verbunden sind, wird die Adressierung der Portexpander unmittelbar mit den Displays verbunden. Somit werden keine weiteren Steuerleitungen zu den Displays benötigt.

Auf Grund der maximalen Baudrate von 8 Mega Bit pro Sekunde (Mbps) des SPI bedingt durch die verwendeten Displays und der Standard Baudrate von 100 Kilo Bits pro Sekunde (kBps) des I²C ergibt sich durch dieses Design des Busses ein Engpass für die Kommunikation mit den Displays. Die Framerate bezeichnet wie oft Bilder an das Display innerhalb einer Sekunde übertragen werden können. Sie wird in Frames pro Sekunde (Fps) angegeben. Die stark begrenzte Framerate berechnet sich anhand von Gleichung 6.1:

$$\text{Framerate}[\text{Fps}] = \frac{\text{Geschwindigkeit des Busses}[\text{Bps}]}{\text{Datenmenge}(\text{Bild})[\text{BpF}] \cdot \text{Anzahl der Teilnehmer}} \quad \text{Gl. (6.1)}$$

Die Datenmenge pro Bild berechnet sich nach Gleichung 6.2 wie folgt:

$$\text{Datenmenge}(\text{Bild})[\text{BpF}] = \frac{\text{Bits per Pixel}[\text{Bit}] \cdot \text{Zeilen} \cdot \text{Spalten} + \text{Bits zur Adressierung}[\text{Bit}] \cdot \text{Adressierungen}}{1[\text{Frames}]} \quad \text{Gl. (6.2)}$$

Durch Einsetzen von Gleichung 6.2 in Gleichung 6.1 ergibt sich Gleichung 6.3 welche der zur Berechnung, des mit dem konfigurierten SPI-Bus möglichen Framerate, zu Grunde liegt

$$\text{Framerate} = \frac{\text{Geschwindigkeit des Busses}[\text{Bps}]}{\left(\frac{\text{Bits per Pixel}[\text{Bit}] \cdot \text{Zeilen} \cdot \text{Spalten} + \text{Bits zur Adressierung}[\text{Bit}] \cdot \text{Adressierungen}}{1[\text{Frames}]} \right) \cdot \text{Anzahl der Teilnehmer}} \quad \text{Gl. (6.3)}$$

Die Berechnung wird nun in Gleichung 6.4 vorgenommen:

$$\text{Framerate}[\text{Fps}] = \frac{8000000[\text{Bps}]}{\left(\frac{16[\text{Bit}] \cdot 160 \cdot 128}{\frac{16[\text{Bit}] \cdot 160}{1[\text{Frames}]}} \right) \cdot 8} \approx 3,0281 [\text{Fps}] \quad \text{Gl. (6.4)}$$

Es muss nun noch die Verzögerung für das Setzen des Chip Select berücksichtigt werden. Die Übertragung des Signals über den I²C-Bus an den MCP23017 verbraucht zwar eine relativ kurze Zeit zwischen den Einzelbildern, sie beeinflusst allerdings die Bildwiederholrate negativ. Die untere Gleichung 6.5 berechnet die Verzögerung pro Frame:

$$\text{Steuerung}[\text{Fps}] = 1[\text{Frames}] \cdot \left(\frac{\text{Bit des CS-Signal}[\text{Bit}] \cdot \text{Anzahl der Signale} \cdot \text{Anzahl der Teilnehmer}}{\text{Geschwindigkeit des Busses}[\text{Bps}]} \right) \quad \text{Gl. (6.5)}$$

Die Berechnung wird in Gleichung 6.6 vorgeführt.

$$\text{Steuerung}[\text{Fps}] = 1[\text{Frames}] \cdot \left(\frac{19[\text{Bit}] \cdot 2 \cdot 8}{100000[\text{Bps}]} \right) = 0,00304 [\text{Fps}] \quad \text{Gl. (6.6)}$$

Die Berechnung der Übertragung inklusive der Übertragung der Steuersignale wird in Gleichung 6.7 vorgenommen:

$$\begin{aligned} \text{Gesamt_Framerate}[\text{Fps}] &= \text{Framerate}[\text{Fps}] - \text{Steuerung}[\text{Fps}] \\ &\approx 3,0281[\text{Fps}] - 0,00304[\text{Fps}] \approx \underline{\underline{3,0251[\text{Fps}]}} \end{aligned} \quad \text{Gl. (6.7)}$$

Somit ergibt sich nach Gleichung 6.7 eine maximale Bildwiederholrate von 3,0251 Einzelbildern pro Sekunde, sicherheitshalber werden jedoch 3 Frames pro Sekunde bei acht Busteilnehmern angenommen. Bei dem vorliegenden Gameport mit lediglich vier Anschlüssen und somit vier Busteilnehmern kann theoretisch die Framerate auf bis zu 6 Frames pro Sekunde heraufgesetzt werden.

Der Gameport selbst wird nicht als Gerät im Ordner `/dev` angezeigt. Dies ist an dieser Stelle nicht nötig und liefert keine Funktion. Da das Gerät nur zur Initialisierung, Verwaltung und zum Anlegen der weiteren Gerätestrukturen dient wird keine weitere Funktionalität nach außen zur Verfügung gestellt.

Um den Bus voll parallel zu halten wurden die nicht gemeinsam nutzbaren Pins als Funktionen der MCP23017-ICs ausgelegt. Eine als weiterhin gemeinsam nutzbare Funktion blieb der Hardware Reset der am Bus befindlichen Geräte. Diesen teilen sich alle Peripheriegeräte.

Auf die Belegung der Bus-Schnittstelle wurde ausführlich in Kapitel 2.3 Belegung der Busschnittstelle eingegangen. Wie bereits dort erwähnt wurde versucht die Anzahl der Kommunikationsleitungen möglichst minimal zu halten.

6.4.1 Platform_Data zum Informationsaustausch

Die Treiber wurden derart gestaltet, dass sie die benötigten Konfigurationen jeweils von

der nächst höheren Konfigurationsebene erhalten. Die Treiber reichen die Konfiguration mittels des Platform_Data-Konzepts an die nächste Treiberebene durch.

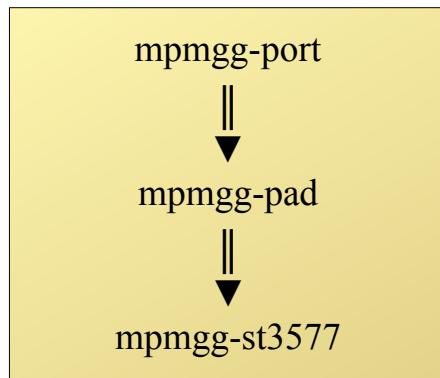


Abbildung 6.2: Treiberebenen für die Geräte

Die für eine korrekte Konfiguration verwendeten Treiber-Ebenen sind in der oberen Abbildung 6.2 dargestellt. Jede Ebene generiert Informationen, welche von einer tieferliegenden Ebene benutzt werden. Mittels der Platform_Data-Pointer werden diese Informationen zwischen den Ebenen ausgetauscht. Auf diese Weise werden den einzelnen Geräten zentral generierte Informationen zur Verfügung gestellt. In „mpmgg-port“ werden, wie im folgenden Abschnitt 6.6.1 genauer beschrieben, die benötigten Pointer zu den Spinlocks erstellt, welche zum sicheren Übertragen der Datenpakete über die Bus-Schnittstellen verwendet werden. Diese werden von „mpmgg-pad“ und „mpmgg-st7735“ verwendet. Zudem werden die Geräteidentifikationen der „mpmgg-st7735“-Platform_Device generiert und mittels dem entsprechenden „mpmgg-pad“-Platform_Device weiter an „mpmgg-st7735“ gereicht. Der Treiber „mpmgg-pad“ verfügt über genauere Informationen für den Zugriff auf die Funktionen des MCP23017. Diese Informationen werden an die „mpmgg-st7735“-Platform_Device weiter gereicht. Genaue Informationen zu dem Einsatz des Platform_Data-Konzepts können den Kapiteln 6.6.1 Implementierung des Gameport-Treibers „mpmgg-port“, 6.6.2 Implementierung des Game-Pad-Treibers „mpmgg-pad“ und 6.6.3 Implementierung des Display-Treibers „mpmgg-st7735“ entnommen werden.

6.4.2 Variablen mit static Deklaration

Auf eine Variablen Deklaration mit dem Codewort „static“ wurde weitestgehend verzichtet. Selbstverständlich hätte dieses Konzept ebenfalls eingesetzt werden können um die Sicherung der Buszugriffe zu verwalten. Allerdings bietet das bereits genannte Platform_Data-Konzept eine ähnliche Funktionalität welche mittels Pointern flexibler genutzt werden kann.

Eine jedoch wesentliche Stelle des Einsatzes dieses Konzeptes ist die von den Displays gemeinsam verwendete Anmeldung als SPI-Gerät. Die für den SPI-Zugriff verwendete gemeinsam genutzte Datenstruktur wird vom Display-Treiber erzeugt und mittels static-Pointer den angemeldeten Datenstrukturen der identischen Displays zur Verfügung gestellt.

6.5 Framebuffer Schnittstelle

Ein Framebuffer Gerät wird vom System verwendet um grafische Ausgaben von einer im System standardisierten Schnittstelle an ein grafisches Ausgabegerät weiter zu leiten. Hierfür stellt der Geräte-Treiber einen Speicherbereich des Kernels zur Verfügung. Der Kernel selbst verfügt weiter über Funktionen zur Manipulation dieses Speicherbereichs. Das Kernelmodul erweitert diese Fähigkeiten um die von der Hardware benötigten speziellen Zugriffsschnittstelle, welche bei Aktionen auf dessen Speicherbereich verwendet wird.

Im vorliegendem Treiber wird der Framebuffer vom Kernelmodul „mpmrg-st7735“ am System angemeldet, initialisiert und verwaltet. Der Treiber ist nahe an der Vorlage des „fb_st7735“ [I12] ausgerichtet. Dieses Kernelmodul ist bereits Teil der meisten Distributionen für den Raspberry Pi 3 und kann beim Kompilieren des Kernels auch voll integriert werden. Während der „fb_st7735“ direkt GPIOs des Raspberry Pi nutzt um die Steuerleitungen des Displaymoduls zu bedienen, verwendet der „mpmrg-st7735“ Teile der Treiberdstruktur des „mpmrg-pad“ um die Beschaltung der Steuerleitungen über den MCP23017 zu ermöglichen.

Die vom System stammenden Grundfunktionalitäten des Framebuffer werden vom „mpmrg-st7735“-Treiber verwendet um die Kommunikation zwischen dem Systemspeicher und den Registern des ST7735, welches auf dem Display-Modul eingesetzt wird, mittels SPI-Bus zu verwalten. Initialisierung und Verwendung des vom ST7735 genutzten Protokolls wurden dem „fb_st7735“ entnommen und auf die Besonderheiten des vorliegenden Systems angepasst.

6.6 Implementierung

Die folgenden Abschnitte beschreiben die notwendigen Schritte zur Implementierung der Gerätetreiber „mpmrg-port“, „mpmrg-pad“ und „mpmrg-st7735“. Ein strukturelle Beschreibung der umgesetzten Verhaltensweisen und der bei der Entwicklung auftretenden Probleme wird detailliert ausgeführt. Für eine Beschreibung der Codezeilen wird auf den Code selbst und die Kommentare im Code auf der beiliegenden DVD verwiesen.

6.6.1 Implementierung des Gameport-Treibers „mpmrg-port“

Der Gameport stellt eine der drei logischen Abstraktionsebenen der Treiber-Initialisierungen da. Die Aufgaben der zugrundeliegenden Hardware sind zunächst passiver Natur, wurden jedoch in der späten Entwicklungsphase um eine parallele Initialisierung der Displays erweitert. Die Platine, welche direkt auf den Raspberry Pi 3 gesteckt wird, stellt ein logisches Bussystem aus den Pins des Raspberry Pi 3 GPIO Header und der zusätzlich auf der Platine angebrachten, vom Step-Down Konverter stammenden Stromversorgung zusammen und verzweigt diese an die 2x5 IDC-Schnittstellen. Weitere Informationen zur Belegung der 2x5 IDC-Schnittstelle können Kapitel 2.3 Belegung der Busschnittstelle entnommen werden. Der Treiber dient zum Anlegen der am Bus befindlichen Geräte.

Beim Initialisieren des Gameport-Geräts wird ein Hardwarereset aller am Port

befindlichen Peripheriegeräte ausgeführt. Hierfür greift der Treiber auf die GPIOs des Raspberry Pi 3 zu und verändert den Pegel des zuständigen Pins (GPIO 17). Der Reset erfolgt indem der Pin zunächst 20 Millisekunden auf LOW (GND) und danach für den gesamten betrieb des Treibers auf HIGH (3v3) geschaltet wird.

Der nächste Schritt ist der Initialisierung des Gameport-Geräts ist die Bereitstellung der erforderlichen Spinlocks für die Datenbusse. Es wird sowohl ein Spinlock für den Datentransfer des SPI-Busses wie auch für den Datentransfer des I²C-Bus angelegt. Diese sichern die Nutzung der kritischen Abschnitte der entsprechenden nachfolgenden Treiber gegen Scheduler bedingte Wechsel der Daten während einer Übertragung. Auf diese Weise wird sicher gestellt, dass die Übertragungen auf den Bussen immer vollständig, in korrekter Reihenfolge und ohne Datenverlust vorgenommen werden.

Wie bereits erwähnt wurde die Initialisierung der Displays im Laufe der Entwicklung vollständig in die Initialisierung des Gameports eingebettet. Dieser Schritt war notwendig um einen kritischen Timingfehler bei der Initialisierung von mehr als 2 Displays zu korrigieren. Zudem verkürzt dieses Vorgehen die Zeit bis die Geräte am Bus zur Verfügung stehen. Die parallele Konfiguration der entsprechenden Register auf den Displaymodulen wird durch die identische Konfiguration aller Displays ermöglicht. Genaue Informationen zu den Anpassungen, welche hier vorgenommen wurden, sind Abschnitt 6.7.1 Parallele Initialisierung der Displays zu entnehmen.

Die vorbereiteten Spinlocks werden im Anschluss beim Anlegen der Game-Pads als I²C-Gerät im Kernel mittels Platform_Data an die entsprechenden Gerätstrukturen übergeben. Die weitere Prozessierung geschieht mittels des „mpmrgg-pad“-Treibers.

6.6.2 Implementierung des Game-Pad-Treibers „mpmrgg-pad“

Das Game-Pad ist die zweite zu initialisierende Instanz der benötigten Treiberstruktur. Der zugehörige Gerätetreiber „mpmrgg-pad“ meldet Stellt die logische Komponente des Game-Pads dar. Nachdem „mpmrgg-port“ das I²C-Gerät am I²C-Bus angemeldet hat beginnt die Initialisierung und die benötigte Anmeldung am System.

Vor der Anmeldung am System wird das Gerät mit der zugehörigen Busadresse auf Erreichbarkeit geprüft. Hierfür wird der I²C Chip MCP23017 zunächst konfiguriert und im Anschluss ausgelesen. Wenn der MCP23017 korrekt antwortet wird die Anmeldung des Eingabegerätes vorgenommen. Das am System anzumeldende Eingabegerät benötigt umfangreiche Informationen bezüglich der zur Verfügung stehenden Systemeingaben. Diese wurden dem „USB SNES Game-Pad/Controller for PC“ von iNNEXT® mittels Reverseengineering nachempfunden. Die an das System gemeldeten Eingaben, die Benennung innerhalb des Treibers so wie die Maskierung am MCP23017 können der folgenden Tabelle 6.3 entnommen werden.

Tabelle 6.3: Tastenbelegung

Systemeingabe	ID	Registername	Belegung am MCP23017 Port	Bit
ABS_X	1	MPMGG_KEY_INP_UP	B	0x80
	2	MPMGG_KEY_INP_DOWN		0x10
ABS_Y	3	MPMGG_KEY_INP_LEFT	B	0x20
	4	MPMGG_KEY_INP_RIGHT		0x40
BTN_TOP2	5	MPMGG_KEY_INP_TL	B	0x08
BTN_BASE3	6	MPMGG_KEY_INP_SELECT	B	0x04
BTN_BASE4	7	MPMGG_KEY_INP_START	A	0x20
BTN_PINKIE	8	MPMGG_KEY_INP_TR	A	0x10
BTN_TRIGGER	9	MPMGG_KEY_INP_X	A	0x04
BTN_TOP	10	MPMGG_KEY_INP_Y	A	0x01
BTN_THUMB	11	MPMGG_KEY_INP_A	A	0x08
BTN_THUMB2	12	MPMGG_KEY_INP_B	A	0x02

Die folgende Abbildung 6.3 zeigt die Tasten des Game-Pad, sie können anhand der Spalte ID aus Tabelle 6.3 zugeordnet werden.

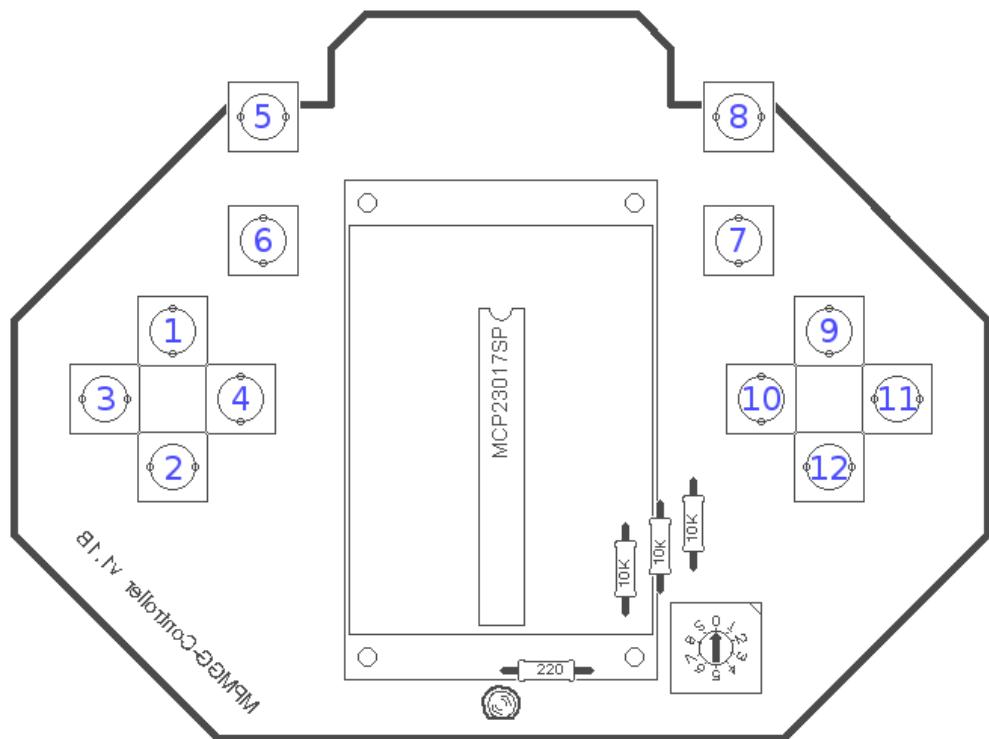


Abbildung 6.3: Tastenbelegung des Game-Pads

Die dem System mitgeteilten Eingabemöglichkeiten werden nach der Anmeldung am System vom Betriebssystem registriert. Der Treiber muss hierfür die angemeldeten Eingaben dem System mitteilen. Dies geschieht intern über Bibliotheken des Betriebssystems. Ebenfalls aus dieser Bibliothek stammt die Registrierungs-Funktion mit der sich der Treiber als Eingabegerät am Betriebssystem anmeldet.

Nachdem das Eingabegerät am System angemeldet ist startet das „mpmgg-pad“-Kernelmodul die Initialisierung des „mpmgg-st7735“-Kernelmoduls. Die hierfür notwendigen Daten werden zusammengestellt und mittels Plattform-Device an den Treiber übergeben.

Der letzte Schritt bei der Initialisierung des Game-Pad ist das Starten eines Arbeits-Thread. Dieser Thread wird regelmäßig alle 30 Millisekunden aufgerufen. Jedes initialisierte Game-Pad besitzt seinen eigenen Arbeits-Thread mit den Informationen des zugehörigen MCP23017 am I²C-Bus. Diese Threads laufen asynchron voneinander und werden mittels Mutex voreinander gesichert. Da die Game-Pads ihre Eingaben nicht mittels Interrupt melden können, fragen die zuvor genannten Threads regelmäßig die Eingaben der Controller ab. Diese Eingaben werden interpretiert und an das System gemeldet.

Aufgrund hoher Bitfehlerraten bei der Verwendung mehrerer Game-Pads wurden in der späten Entwicklungsphase die Polling-Funktionen um Maßnahmen der Signalbereinigung ergänzt. Mehr Informationen hierzu können Abschnitt 6.7.2 Anpassungen zur Verminderung von Bitfehlern auf dem I²C-Bus entnommen werden.

6.6.3 Implementierung des Display-Treibers „mpmgg-st7735“

Die Aufgabe des „mpmgg-st7735“-Treibers ist das korrekte Initialisieren des Displays, gefolgt vom Anlegen der benötigten Systemschnittstellen. Wie in Abschnitt 6.7.1 Parallele Initialisierung der Displays wurde die Initialisierung der Displayperipherie in der fortgeschrittenen Entwicklung in die Initialisierung des „mpmgg-port“-Treibers verlagert um einen schwerwiegenden Timingfehler zu beseitigen und die Verwendung von mehr als zwei Displays zu ermöglichen. Dennoch soll an dieser Stelle die Initialisierungssequenz, welche aus dem zugrundeliegendem Treiber „fb_st7735“ entnommen wurde, beschrieben werden.

Die Initialisierungssequenz besteht aus mehreren Befehlsaufrufen und dem Schreiben in diverse Register des ST7735. Die Funktionalität dieser Aufrufe werden im Folgendem Abschnitt beschrieben.

Tabelle 6.4: Registerkonfiguration des ST7735

Reihenfolge:	Registername:	Registeradresse:	Zielwert:
1	FRMCTR1	0xB1	0x012C2D
2	FRMCTR2	0xB2	0x012C2D
3	FRMCTR3	0xB3	0x012C2D012C2D
4	INVCTR	0xB6	0x07
5	PWCTR1	0xC0	0xA20284
6	PWCTR2	0xC1	0xC5
7	PWCTR3	0xC2	0x0A00
8	PWCTR4	0xC3	0x8A2A
9	PWCTR5	0xC4	0x8AEE
10	VMCTR1	0xC5	0x0E
11	COLMOD	0x3A	0x05

Die Initialisierung der ST7735-Peripherie geschieht mit dem Aufruf des Befehls SWRESET (0x01). Dies führt den Software-Reset des Display-Controllers aus. Im Anschluss folgt der Befehl SLOPOUT (0x11) welcher den Sleep-Modus deaktiviert und den Ausgabeboost des Display-Controllers aktiviert. Der nächste Schritt ist das Beschreiben diverser Register. Die Reihenfolge und die entsprechenden Zielwerte können Tabelle 6.4: Registerkonfiguration des ST7735 entnommen werden. Es folgt nunmehr die reine Beschreibung der Funktionalität dieser Werte. Register FRMCTR1 definiert das Framerateverhalten im normalen Modus. Diese berechnet sich wie in Gleichung 6.8.

$$\text{Framerate}_{\text{normal}}[\text{Fps}] = \frac{f_{osc}}{1 \times 2 + 40} \cdot (\text{LINE} + 2C + 2D) \quad \text{Gl. (6.8)}$$

Register FRMCTR2 definiert das Framerate verhalten im Modus Leerlauf, diese berechnet sich wie in Gleichung 6.9.

$$\text{Framerate}_{\text{idle}}[\text{Fps}] = \frac{f_{osc}}{1 \times 2 + 40} \cdot (\text{LINE} + 2C + 2D) \quad \text{Gl. (6.9)}$$

Das Register FRMCTR3 setzt Framerate-Controllen. Es werden die folgende Modi für den Modus Normal, wie auch den Modus Leerlauf verwendet:

- partial mode
- dot inversion mode
- line inversion mode

Das Register INVCTR steuert die Inversionskontrolle des Displays, diese wird deaktiviert.

Das Register PWCTR1 steuert das Energieverwaltung des ST7735. PWCTR1 wird wie folgt konfiguriert:

- -4.6V
- AUTO Modus

Das Register PWCTR2 steuert das Energiemanagement des ST7735. PWCTR2 wird wie folgt konfiguriert:

- VGH25 = 2.4C
- VGSEL = -10
- VGH = 3 * AVDD

Das Register PWCTR3 steuert das Energiemanagement des ST7735. PWCTR3 wird wie folgt konfiguriert:

- OpAmp current small
- Boost frequency

Das Register PWCTR4 steuert das Energiemanagement des ST7735 im normalen Modus. PWCTR4 wird wie folgt konfiguriert:

- BCLK/2

- Opamp current small & Medium low

Das Register PWCTR5 steuert das Energiemanagement des ST7735 im Leerlauf Modus. PWCTR5 wird wie folgt konfiguriert:

- Opamp current small & Medium low

Das Register VMCTR1 steuert das Energiemanagement im EPROM des ST7735. VMCTR1 definiert die Erkennung von Schwellwerten für HIGH und LOW Signalen und wird in der Konfiguration deaktiviert.

Das Register COLMOD steuert das Farbverhalten des ST7735, PWCTR4 wird wie folgt konfiguriert:

- 16 Bit pro Pixel

Der nächste Schritt ist das Ausführen des Befehls INVOFF (0x20). Dieser deaktiviert die Display-Inversion. Mit dem Befehl DISPON (0x29) wird zum Schluss des Display Aktiviert und die Initialisierung abgeschlossen.

Nach der Initialisierung des Display-Moduls meldet der „mpmgg-st7735“-Treiber das Gerät am System an. Hierfür wird das Gerät mittels framebuffer_alloc am System angemeldet. Die Gerätedatei wird hierbei automatisch erstellt und ergibt sich aus dem Schema „fb“ gefolgt von einer laufenden Nummer. Die fb_info Struktur, welche mittels Pointer zurückgegeben wird, muss mit weiteren Informationen versehen werden.

Die folgende Tabelle 6.5 beschreibt die Zugriffsfunktionen des Treiber und wie diese verwendet wurden. Die hierfür zu füllende Struktur ist als fbops Pointer innerhalb der Struktur angelegt, der Pfad lautet somit fb_info→fbops.

Tabelle 6.5: Dateizugriffe des „mpmgg-st7735“-Treibers

Pfad	Zugewiesene Funktion	Beschreibung
fb_info→fbops→ fb_read	fb_sys_read	Ermöglicht das Lesen des Videospeichers aus der Gerätedatei
fb_info→fbops→ fb_write	st7735r_fb_write	Ermöglicht das Schreiben des Videospeichers in die Gerätedatei
fb_info→fbops→ fb_fillrect	st7735r_fb_fillrect	Schreibt ein gefülltes Rechteck in den Videospeicher
fb_info→fbops→ fb_copyarea	st7735r_fb_copyarea	Kopiert einen Bereich des Videospeichers
fb_info→fbops→ fb_imageblit	st7735r_fb_imageblit	Schreibt ein Bild in den Videospeicher
fb_info→fbops→ fb_setcolreg	st7735r_fb_setcolreg	Farbkorrektur des zu schreibendem Bildes auf ein für die Zielhardware verständliches Farbschema
fb_info→fbops→ fb_blank	st7735r_fb_blank	Leert den Videospeicher

Die folgende Tabelle 6.6 beschreibt die Variablenbelegung der Bildschirmeinstellungen des Treiber. Die hier gefüllten Strukturen sollten nicht zur Laufzeit geändert werden um

die Funktionalität nicht zu stören.

Tabelle 6.6: Einstellungen zur Auflösung des „mpmrg-st7735“-Treibers

Parameter	Wert	Beschreibung
info→fix. type	FB_TYPE_PACKED_PIXELS	Typ der Speicherorganisation
info→fix. visual	FB_VISUAL_TRUECOLOR	Darstellbare Farben
info→fix. xpanstep	0	Kein Hardwarepanning
info→fix. ypanstep	0	Kein Hardwarepanning
info→fix. ywrapstep	0	Kein Hardware-ywrap
info→fix. accel	FB_ACCEL_NONE	Hardwarebeschleunigung (deaktiviert)
info→fix. smem_len	40960	Größe des Videospeichers $\frac{xres \cdot yres \cdot bits_per_pixel}{8}$
info→var. rotate	0	Rotation des Bildschirms
info→var. xres_virtual	128	Bildschirmauflösung (Breite)
info→var. yres_virtual	160	Bildschirmauflösung (Höhe)
info→var. bits_per_pixel	16	Bits welche für ein Pixel
info→var. nonstd	1	Kein Standard Pixelformat
Info→ flags	FBINFO_FLAG_DEFAULT FBINFO_VIRTFB	Weiter Informationen zum Treiber

Die folgende Tabelle 6.7 beschreibt die Variablenbelegung der Farbeinstellung pro Pixel des Treiber. Die hier gefüllten Strukturen sollten nicht zur Laufzeit geändert werden um die Funktionalität nicht zu stören. Die Farbwerte werden in einem 16 Bit Speicher abgelegt. Die Farbaufteilung wird in der unteren Abbildung 6.4 aus Quelle [I14] illustriert.

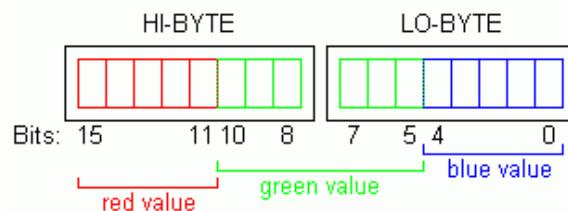


Abbildung 6.4: 16-Bit Farbaufteilung pro Pixel

Tabelle 6.7: Einstellungen zur Auflösung des „mpmrgg-st7735“-Treibers

Parameter	Wert	Beschreibung
info→var. red.offset	11	Verschiebung des Rot-Kanals
Info→var. red.length	5	Anzahl der Bits für den Rot-Kanal
info→var. green.offset	5	Verschiebung des Grün-Kanals
info→var. green.length	6	Anzahl der Bits für den Grün-Kanal
info→var. blue.offset	0	Verschiebung des Blau-Kanals
info→var. blue.length	5	Anzahl der Bits für den Blau-Kanal
info→var. transp.offset	0	Verschiebung des Alpha-Kanals
info→var. transp.length	0	Anzahl der Bits für den Alpha-Kanal

6.7 Anpassungen nach dem Test mit mehreren Game-Pads

Nach dem der größte Teil der Entwicklung mit einem einzigen Display vorgenommen wurde wurden weitere Einheiten des Game-Pads gefertigt. Diese wurden aus kommerziell hergestellten Platinen durch Handbestückung erstellt. Bei den Tests mit mehreren Game-Pads, welche den gemeinsamen Bus verwenden, wurden einige Fehler aufgedeckt. Die hieraus entstandenen Anpassungen werden in den nachfolgenden Kapiteln beschrieben.

6.7.1 Parallelle Initialisierung der Displays

Die Displaymodule benötigen vor dem ersten Betrieb eine Konfigurationssequenz. Dies führte bei der Verwendung mehrerer Game-Pads an der Bus-Schnittstelle zu Timingfehlern. Wenn über den vom Raspberry Pi ausgeführten Reset-Pin alle Geräte am Bus zeitgleich zurückgesetzt werden müssen alle Geräte neu konfiguriert werden. Ein gezieltes Schalten eines Reset-Signals von nur einem Bus-Teilnehmer ist nicht vorgesehen. Mit der Konfiguration der Displays muss allerdings umgehend nach dem Reset begonnen werden. Das Konfigurieren mehrerer Displays nacheinander nimmt mehr als die zur Verfügung stehende Zeit in Anspruch. Es wurde eine maximale Anzahl von sequenziell konfigurierbaren Displays von zwei per Experiment ermittelt. Um dieses Problem zu vermeiden wurde die Initialisierung aus dem „mpmrgg-st7735“-Kernelmodul in eine parallele Variante umgeschrieben und in das „mpmrgg-port“-Kernelmodul eingebettet.

Die SPI-Kommunikation wird ausschließlich vom Raspberry Pi 3 hin zu dem Displaymodul vorgenommen. Es wird weder während der Initialisierung noch im späteren Betrieb ein Rücksignal oder eine Antwort der Displays erwartet. Alle

Busteilnehmer mit Ausnahme des Raspberry Pi verhalten sich passiv und lesen ausschließlich vom SPI-Bus, daher können synchron auf alle Displays übertragen werden.

Die Grundvoraussetzung damit die Displaymodule initialisiert werden können ist die korrekte Beschaltung der Steuerleitungen der Displaymodule mit den entsprechenden Steuersignalen. Die Steuersignale müssen adressiert über den I²C-Bus an die MCP23017 Portexpander übertragen werden. Die MCP23017 verfügen über eine einfache Registeraufteilung und Flags zur Steuerung welche dem Adressbyte hinzugefügt werden. Die untere Abbildung 6.5 von [I15] illustriert die erste an einen MCP23017-IC zu sendende Bitsequenz. Startbit und ACK-Bit gehören zum I²C-Busprotokoll. Der Raspberry Pi tritt am I²C-Bus als Busmaster auf. Alle MCP23017 sind Slave-Geräte am I²C-Bus. Das auf Abbildung 6.5 zu erkennende ACK-Bit dient zur Meldung einer erfolgreichen Übertragung vom Master- zum Slave-Gerät und wird entsprechend vom Slave-Gerät gesetzt. Wird dieses Flag nicht gesetzt, so handelt es sich um einen Übertragungsfehler.

FIGURE 3-4: I²C CONTROL BYTE FORMAT

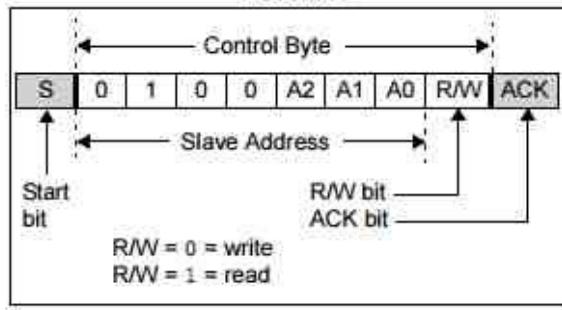


Abbildung 6.5: I²C-Adressbyte des MCP23017

Das Setzen der Steuerbits jedes Displaymoduls am zugehörigen MCP23017 muss also nacheinander geschehen. Die MCP23017 müssen zunächst konfiguriert werden. Dies geschieht wie in Abschnitt 6.3 Zugriff auf MCP23017 beschrieben. Die Registerkonfiguration wird sequenziell für alle acht potentiell am Bus befindlichen Portexpander vorgenommen. Wird die Konfiguration eines nicht am I²C-Bus befindlichen Portexpanders geschrieben, so gibt die I²C-Systemschnittstelle einen Fehlercode zurück. Dieser Fehlercode wird nicht berücksichtigt und die Konfiguration weiter vorgenommen.

Die Übertragung einer von Steuersignalen eingegrenzten Konfigurationsbytefolge an alle Displaymodule wird nun ermöglicht, indem alle acht potentiell am Bus befindlichen Portexpander nacheinander mit den entsprechenden Steuerbits konfiguriert werden und im Anschluss die Bytefolge mittels SPI-Bus übertragen wird. Werden die Register eines nicht am I²C-Bus befindlichen Portexpanders geschrieben, so gibt die I²C-Systemschnittstelle einen Fehlercode zurück. Dieser Fehlercode wird nicht berücksichtigt; es wird mit der Konfiguration des nächsten Portexpanders fortgefahrene.

Die Konfiguration wird wie in Kapitel 6.6.3 Implementierung des Display-Treibers „mpmogg-st7735“ anhand Tabelle 6.4 Registerkonfiguration des ST7735 beschrieben

vorgenommen. Die Anmeldung der Eingabegeräte und der Framebuffer wird weiterhin in den entsprechenden Kernelmodulen „mpmgg-pad“ und „mpmgg-st7735“ vorgenommen.

6.7.2 Anpassungen zur Verminderung von Bitfehlern auf dem I²C-Bus

Durch den Test mit mehreren Game-Pads am gemeinsamen Bus wurde eine starke Einstreuung des SPI- auf den I²C-Bus festgestellt. Dies erzeugt eine hohe Fehlerrate. Da die Übertragungsfehler weder mit einem Korrekturmecanismus versehen sind, noch Bitfehler auf der Empfängerseite direkt detektierbar sind, müssen Maßnahmen getroffen werden um Fehleingaben nicht direkt via der am System angemeldeten Gerätedatei mit zu teilen.

Hierfür wurde die Funktion *poll_Ops* des Kernelmoduls „mpmgg-pad“, welche für das Pollen der Tasteneingaben vom Game-Pad zuständig ist, um ein zweistufiges Konzept zur Fehlerkorrektur und Fehlervermeidung erweitert.

Als erste Maßnahme zur Filterung von Fehleingaben wurde eine Mehrfachabfrage der MCP23017-ICs eingeführt. Der Portexpander wird drei mal nach den Werten in den Registern GPIOA und GPIOB abgefragt. Die beiden Register werden in einem Integer zusammengesetzt, die drei Abfragen werden bitweise verundet um einzelne Signalspitzen ab zu fangen. Signalspitzen sind eine Folge der langen Signalwege in den parallel verlaufenden Leitungen und den Induktionseffekten bedingt durch die Signalwechsel. Auf Grund der unterschiedlichen Übertragungsgeschwindigkeiten der beiden Bussysteme ist zu erwarten, dass die Signalspitzen nicht regelmäßig eingestreut werden. Die Einstreuung von LOW-Werten ist kaum zu erwarten. Sie senken den Signalpegel, allerdings bleibt dieser über der Schaltspannung des Transistors im Raspberry Pi. Durch die dreifache Abfrage und die Verundung wird der niedrigste gelesene Einzelbitzustand als Faktum angenommen. Das Ergebnis dieser Verarbeitung wird zur zweiten Ebene der Sicherung weiter geleitet.

Es wurde festgestellt, dass ein weiterer Fehlertyp eine Übertragung mit besonders vielen HIGH-Signalen ist. Um diesen Fehler zu filtern wird die Kernelfunktion *_sw_hweight16* verwendet. Diese Funktion zählt die HIGH-Bits und liefert die Anzahl als Rückgabewert. Als fehlerhaft wird eine Übertragung gewertet, wenn mehr als 5 Bits den Pegel HIGH besitzen. Diese Grenze wurde festgelegt anhand der maximal sinnvoll zusammen drückbaren Tasten. Es wird angenommen, dass eine Kombination wie zum Beispiel die beiden Top-Trigger an der Stirnseite des Game-Pads, eine Richtungstaste und zwei Funktionstasten der maximalen Anzahl an gleichzeitig sinnvoll zu drückenden Tasten entspricht.

Die auf diese Weise gefilterten Daten werden wie gewohnt interpretiert und an die Kernelschnittstelle der entsprechenden Gerätedatei gemeldet. Das Eingabeverhalten wird durch diese Techniken spürbar verbessert.

7 Test des Treibers

Die folgenden Abschnitte behandeln die Methoden welche für die abschließenden Tests der Peripherie und der zugehörigen Treiber verwendet wurden. Auf diese Weise wurde sichergestellt, dass die Implementierung der vom System geforderten Spezifikation entspricht.

7.1 Test des Adapters

Ein Test des Adapters ist unnötig, da dieser nur als oberste Konfigurationsinstanz dient. Für die korrekte Systemanmeldung wird lsmod ausgeführt. In der Ausgabe muss mpmgg-port aufgeführt sein. Die Betrachtung des Dateisystems hilft da korrekte starten zu analysieren. Wie in den folgenden Kapiteln beschrieben, legt dieses Modul die darunterliegende Treiberstrukturen an und initialisiert diese. Dies führt zu neuen Gerätedateien im verzeichnisc */dev*.

7.2 Test des Game-Pads

Ein erstes Indiz ist das Ausführen von lsmod. In der Ausgabe sollte das Game-Pad als mpmgg-pad aufgeführt werden. Diese Prüfung zeigt allerdings nur ob das Gerät erfolgreich im System angelegt wurde. Die Funktionalität wird erst durch die Initialisierung der Datenstruktur aus dem von mpmgg-port übergebenen Platformdevice ermöglicht. Diese löst das Probe-Kommando aus, welches zu Änderungen am System führt.

Das „proben“ legt im Dateibaum des Systems eine Gerätedatei an, welche sich unter */dev/inputs* finden lässt. Die Joysticks lassen sich daran erkennen, dass die Gerätedatei „js“ genannt wird. Abhängig davon wie viele Joysticks bereits am System angemeldet sind wird der Gerätedatei zusätzlich eine fortlaufende Nummer angehangt. Nachdem die Gerätedatei durch erfolgreiches Ausführen des „Probe“-Kommandos angelegt wurde, kann die Kommunikation durch den von der Betriebssystemschnittstelle definierten Kontaktspunkt getestet werden.

Hierfür wird das Paket joystick mittels Paketmanager installiert. Für die Installation muss das folgende Kommando in der Konsole verwendet werden:

```
sudo apt-get install joystick
```

Das in diesem Paket beinhaltete Programm jstest wird nun zusammen mit der Gerätedatei verwendet um eine Textausgabe in der Konsole zu produzieren, welche die angelegte Datenstruktur des Betriebssystems und deren Datenbestand anzeigt. Das folgende Kommando startet die Ausgabe für den ersten am System angemeldeten Joystick:

```
jstest /dev/input/js0
```

Es erscheint die Folgende Ausgabe:

```
Driver version is 2.1.0.  
Joystick (mpmgg-pad) has 2 axes (X, Y)  
and 8 buttons (Trigger, ThumbBtn, ThumbBtn2, TopBtn,  
TopBtn2, PinkieBtn, BaseBtn3, BaseBtn4).  
Testing ... (interrupt to exit)  
Axes: 0: 0 1: 0 Buttons: 0:off 1:off 2:off  
3:off 4:off 5:off 6:off 7:off
```

Durch drücken der Tasten sollten sich nun diese Ausgaben verändern lassen. Besonders zu beachten ist , dass die vier links am Controller in Kreuzform angeordneten Tasten die Achsen bedienen. Das Drücken dieser Tasten führt zu den Extrema der Achsen, welche sich in den positiven wie in den negativen Bereich ausdehnen.

7.3 Test des Framebuffers

Ein erstes Indiz ist das Ausführen von lsmod. In der Ausgabe sollte das Display als mpmgg-st7735 aufgeführt werden. Diese Prüfung zeigt allerdings nur ob das Gerät erfolgreich im System angelegt wurde. Die Funktionalität wird erst durch die Initialisierung der Datenstruktur aus dem von mpmgg-pad übergeben Platformdevice ermöglicht. Diese löst das Probe-Kommando aus, welches zu Änderungen am System führt.

Das „proben“ legt im Dateibaum des System eine Gerätedatei an, diese lässt sich unter /dev finden. Die Framebuffer lassen sich daran erkennen, dass die Gerätedatei „fb“ genannt werden. Abhängig wie viele Framebuffer bereits am System angemeldet sind, wird der Gerätedatei zusätzlich eine fortlaufende Nummer an gehangen. Nachdem die Gerätedatei durch erfolgreiches ausführen des „Probe“-Kommandos angelegt wurde, kann die Kommunikation durch den von der Betriebssystemschnittstelle definierten Kontaktpunkt getestet werden.

Hierfür wird das Paket „Linux framebuffer imageviewer“ mittels Paketmanager installiert. Für die Installation muss das folgende Kommando in der Konsole verwendet werden:

```
sudo apt-get install fbi
```

Das in diesem Paket beinhaltete Programm fbi wird nun verwendet um Bilder auf dem Display an zu zeigen. Die weiterhin benötigten Parameter sind in der Tabelle 7.1 aufgeführt.

Tabelle 7.1: Verwendete Parameter des fbi-Kommandos

Parameter	Wert	Funktion
-d	/dev/fb1	Angabe der Gerätedatei auf welche geschrieben werden soll.
-vt	1	Startet das Kommando auf der virtuellen Konsole 1
-t	2	Definiert die Zeit in Sekunden zwischen Bildwechseln
-noverbose		Unterdrückt die Ausgabe des Dateipfades und der Bildinformationen auf dem Ausgabegerät
-a	/home/rpi/develop/Treiber/test/*	Markiert alle Dateien im Verzeichnis „/home/rpi/develop/Treiber/test“ als Datenquelle

Das auf diese Weise konfigurierte Kommando lautet:

```
sudo fbi -d /dev/fb1 -vt 1 -t 2 -noverbose -a
/home/rpi/develop/Treiber/test/*
```

Beim Ausführen wechselt das gestartete Programm die in dem Verzeichnis angegebenen Bilder auf dem Display. Durch spezielle Bilder lassen sich spezielle Eigenschaften wie die Konfiguration des Display Random-Access Memory (DRAM) testen. Anfängliche Probleme, hervorgerufen durch einen verschobenen Speicherbereich innerhalb des DRAM, konnten auf diese Weise identifiziert und innerhalb des Treibers korrigiert werden.

8 Automatismen

Die folgenden Abschnitte beschreiben die Anpassungen im System, welche notwendig sind, um automatisch Aufgaben im Zusammenhang mit dem entwickelten Systembestandteilen und deren Umgebung zu erfüllen.

8.1 Automatisches Laden der Treiberstruktur beim Starten des Kernels

Die Treiber wurden so gestaltet, dass sie über interne Abhängigkeiten die korrekte Reihenfolge beim starten des „mpmrgg-port“ einhalten. Diese können vom Betriebssystem erkannt und aufgelöst werden. Notwendig hierfür ist, dass die Abhängigkeiten mittels *depmod* erkannt und aufgelöst werden können. Das Werkzeug *depmod* stammt aus dem Paket **kmod** für den Umgang mit den Modulen und ist bei den meisten Distributionen standardmäßig installiert. Dieses Programm muss wie folgt nach der Installation der neuen Kernelmodule einmalig in der Konsole ausgeführt werden.

```
sudo depmod -a
```

Beim Starten des Kernelmoduls „mpmrgg-port“ werden alle benötigten Treiber automatisch mit gestartet und von „mpmrgg-port“ entsprechend konfiguriert. Um das Treibermodul beim Startvorgang des Systems automatisch mit zu starten und auf diese Weise die tieferliegenden Treiber zu konfigurieren, muss in der Datei /etc/rc.local die folgende Zeile hinzufügen werden:

```
modprobe mpmrgg-port
```

Die Datei /etc/rc.local wird beim Systemstart ausgeführt. Es wird kein vorangestelltes sudo-Kommando benötigt wie in der Konsole sonst üblich. Der Aufrufer „systemd“ besitzt Rootrechte, welche beim Erstellen des Prozesses auf die im Skript definierten Aufrufe übergehen.

9 Weitere Aufgaben im Projekt

Neben der Entwicklung der Hardware- und Softwarekomponenten wurden im Rahmen dieser Projektarbeit weitere Aufgaben übernommen. Einige dieser Aufgaben betreffen den verwandten Projektteil der Spieleentwicklung. Dies wird in Kapitel 9.3 Grafiken und Leveledesign für das Spiel ausführlich beschrieben.

Alle im Folgenden beschriebenen Arbeiten und Entscheidungen bezüglich dieser Hardware- und Softwarekomponenten wurden vom Autor dieses Berichts im vollen Umfang übernommen. Auch wenn sie nicht direkt zur Entwicklung gehören sollen sie der Vollständigkeit halber genannt werden.

9.1 Aufbau der Entwicklungshardware auf Basis des Raspberry Pi

Um während der Entwicklung die Hardware portabel zu halten und sie schnell einsatzbereit zu machen wurde eine Möglichkeit gesucht um den Raspberry Pi ohne TV-Gerät zu betreiben. Aus diesem Grund wurde der Raspberry Pi 3 auf das Raspberry Pi Touch Display montiert.

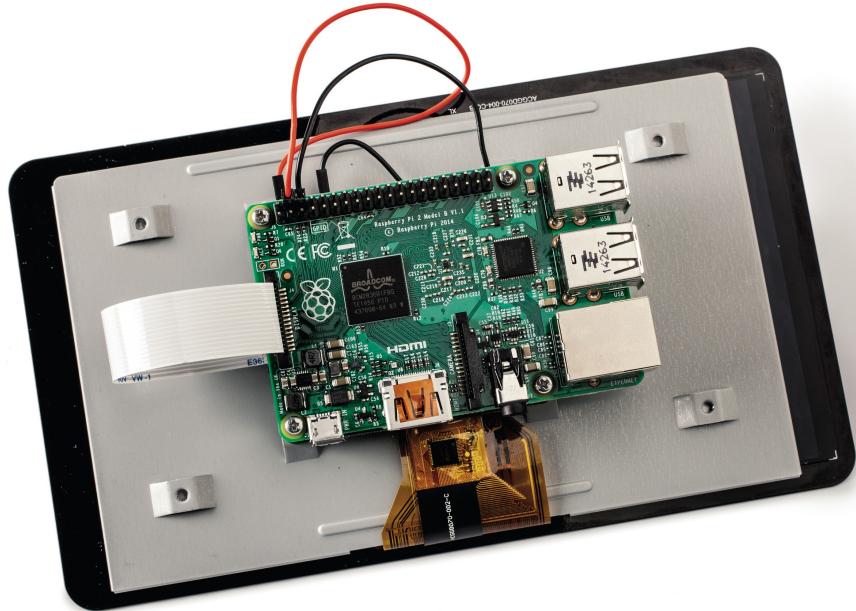


Abbildung 9.1: Rückseite des Raspberry Pi Touch Display mit montiertem RPI

Die obere Abbildung 9.1 aus Quelle [I16] zeigt die Rückseite des Raspberry Pi Touch Display. Der Raspberry Pi ist bereits montiert und angeschlossen. Um die Aufsteckplatine des Portadapters mit dem Raspberry Pi verwenden zu können muss die Stromversorgung separat aus der Aufsteckplatine gewonnen werden. Für diesen Zweck wurde eine spezielle Version des aufsteckbaren Step Down Konverters erstellt, welche die Pins des Step Down Konverters als Pfostenleiste zur Verfügung stellt. Diese ist auch auf Abbildung 4.3 zu sehen. Zu diesen Pins gehört das GND-Netz und das 5 Volt Netz welche beide mit der Hohlbuchse verbunden sind. Die beschriebene Konfiguration kann später verworfen werden. Da der Raspberry Pi 3 als Zielhardware ein TV-Gerät für das Videosignal verwenden soll kann die Standardversion des Step Down Konverters verwendet werden und das Display in diesem Schritt entfernt werden. Die untere Abbildung 9.2 zeigt die finale Hardware-Konfiguration. Der Raspberry Pi 3 wurde auf

einem Sockel montiert, der Portadapter deckt den Raspberry Pi nahezu vollständig ab und einer der Steckplätze für die Game-Pads wurde belegt.

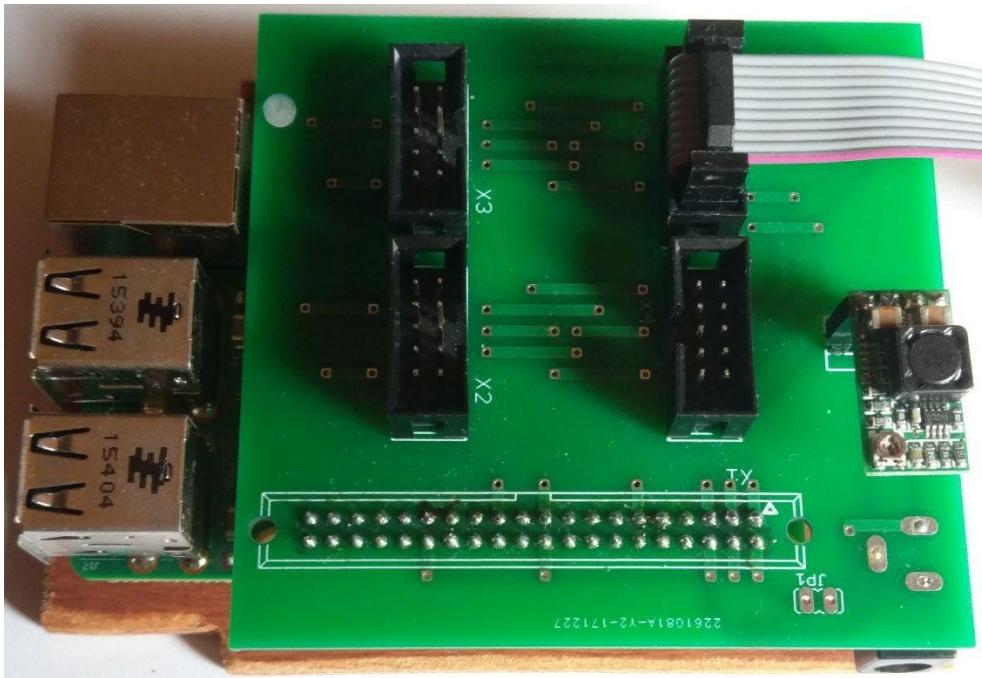


Abbildung 9.2: Finale Konfiguration

Während der Entwicklung wurde ein Ubuntu Betriebssystem eingesetzt. Dies ermöglichte viele Konfigurationen und eine breites Spektrum an kompatiblen Paketen. Die Familie der Ubuntu Betriebssysteme ist als Desktop Betriebssystem entworfen worden. Dies erklärt die Vielzahl an verfügbaren Paketen. Für die Präsentation wurde jedoch ein anderes Betriebssystem favorisiert. Mehr hierzu in Kapitel 9.4 Umzug auf das finale Betriebssystem.

9.2 Konfiguration und Wartung des Raspberry Pi

Um eine netzwerk basierte Entwicklungsumgebung zu schaffen, auf die mittels der in Windows integrierten Dateifreigabe zugegriffen werden kann, wurde der Samba-Server verwendet. Um die entsprechenden Pakete des Samba-Servers zu installieren muss folgendes Kommando ausgeführt werden:

```
sudo apt-get install samba-common samba
```

Nach der Installation der Pakete kann die Datei `/etc/samba/smb.conf` mit dem folgendem Inhalt versehen werden.

```
[Development]
comment = Development
path = /home/pi/Development
writeable = yes
guest ok = yes
create mask = 0644
directory mask = 0755
force user = pi
```

Die auf diese Weise konfigurierte Schnittstelle ermöglicht den Zugriff auf die Ordner /home/pi/Development in welchen sich in weiteren Unterordnern die Dateien des Source Code befinden. In den Unterordnern des Verzeichnisses /home/pi/Development/Treiber befinden sich die Dateien der erstellten Kernelmodule (/home/pi/Development/Treiber/module), Notizen zum testen (/home/pi/Development/Treiber/scripte), so wie einige Testbilder (/home/pi/Development/Treiber/test). Des Weiteren stellt der Unterordner /home/pi/Development/MPMG-Game das Hauptverzeichnis zur Entwicklung des Spiels dar.

Eine weitere Zugriffsvariante, welche benötigt wird Befehle von einem Client-PC auf der Entwicklungsumgebung aus zu führen, ist die Schnittstelle der secure shell (ssh). Diese Schnittstelle kann mit einem entsprechendem ssh-Klienten wie zum Beispiel Putty angesprochen werden. Sie ermöglicht den Netzwerkzugriff auf die Konsole des Kernels.

Der ssh-Server ist auf den meisten Distributionen bereits vorinstalliert. Jedoch muss diese Schnittstelle erst freigegeben werden bevor sie verwendet werden kann. Hierfür muss in der Konsole der Befehl *sudo raspi-config* ausgeführt werden. Die untere Abbildung 9.3 zeigt den sich öffnenden Dialog.[I17]

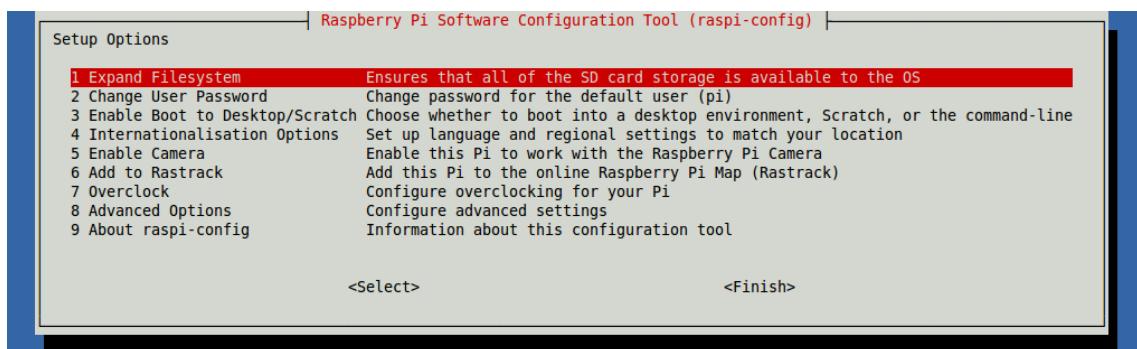


Abbildung 9.3: raspi-config

Unter dem Menü Advanced Options/SSH kann wie in [I18] beschrieben die secure shell aktiviert werden.

9.3 Grafiken und Leveledesign für das Spiel

Während der Projektbesprechung wurde vereinbart, dass die benötigten Spielegrafiken in den Zuständigkeitsbereich dieser Projektarbeit, das Spiel selbst allerdings zu einer separaten Projektarbeit gehören. Dieser Zusammenhang verbindet beide Projektarbeiten, jedoch ermöglicht es das getrennte Bewerten des eigentlichen Spiels

und der verwendeten Hardware. Auch wenn das Erzeugen von Grafiken und das Design von Levels nicht in den Aufgabenbereich eines technisch orientierten Universitätsabschlusses gehören, so waren diese Arbeiten wichtig für die Entwicklung des Spiels und haben Zeit in Anspruch genommen. Daher werden in den folgenden Abschnitten die Grafiken vorgestellt und ihre Verwendung kurz beschrieben.

9.3.1 Grafiken für das Spiel

Für die Elemente des Spiels wurden Grafiken erstellt welche mittels Spriteanimation belebt wurden. Spriteanimation bedeutet, dass eine Bewegung (z.B einer laufenden Spielfigur) mittels mehrerer Einzelbilder, welche den Bewegungsablauf darstellen, durch den schnellen Wechsel der Einzelbilder in Bewegung gesetzt wird. Abbildung 9.4 Zeigt ein Beispiel für die spritebasierte Animation einer Spielfigur.[I22]



Abbildung 9.4: Einzelbilder einer Spriteanimation

Der Grafikstil wurde Retro-Spielen wie Mega Man (1987), Super Mario World (1991) und Sonic the Hedgehog (1991) nachempfunden.[I19][I20][I21] Die Grafiken wurden unter Zuhilfenahme der Bildbearbeitungsprogramme GIMP und Paint erstellt. Die Grafiken besitzen einen Alpha-Layer um den Hintergrund durch die Grafikebenen erkennen zu können. Mehr zu den Grafikebenen in Abschnitt 9.3.2 Leveledesign des Spiels.

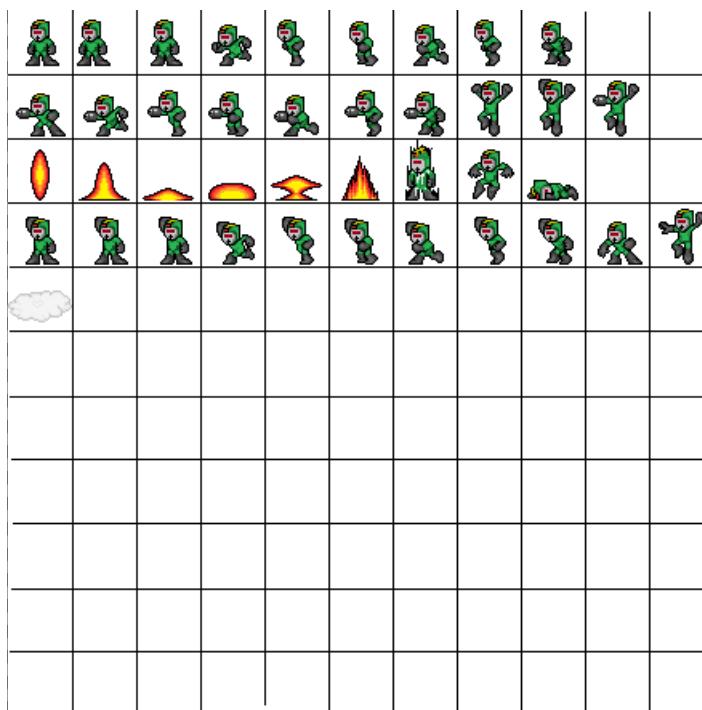


Abbildung 9.5: d-man_work.png

Die obere Abbildung 9.5 zeigt die ersten Grafiken der Spielfigur. Diese wurden zum Testen verwendet. Die Abbildung ist in Originalgröße und ist 440 Pixel breit und 440 Pixel hoch. Anhand der noch vorhandenen Linien zur Abgrenzung der einzelnen Sprites. Betrachtet man die Spritereihen von links nach rechts, dann sind die Bewegungsmuster erkennbar. Die späteren Anforderungen des Spiels ergaben, dass für die bewegten Figuren des Spiels ein gemeinsamer Texturatlas verwendet werden soll, um die Anzahl der *Zeichenaufrufe* zu mindern. Zeichenaufrufe stellen einen Enpass der Verarbeitung dar. Die von der Hardwarebeschleunigung ausgeführten Zeichenaufrufe können sehr effizient verarbeitet werden, problematisch ist allerdings das Laden einzelner Grafiken in den Grafikspeicher. Dies verbraucht viel Zeit und wirkt sich negativ auf die Laufzeit aus. Die Positionierung der Grafikelemente auf dem Display übernehmen die Funktionen der Hardwarebeschleunigung. Aufgrund der begrenzten Leistung wird dringend empfohlen die Hardwarebeschleunigung zu nutzen. Das System on Chip Broadcom BCM2837 1200 MHz 64-Bit ARM Cortex-A53 Quad-Core-Prozessor mit integriertem VideoCore IV Dual-Core-GPU besitzt unter Verwendung des Grafikprozessors (englisch graphics processing unit – GPU) genügend Leistung auch grafiklastige Aufgaben zu verarbeiten.

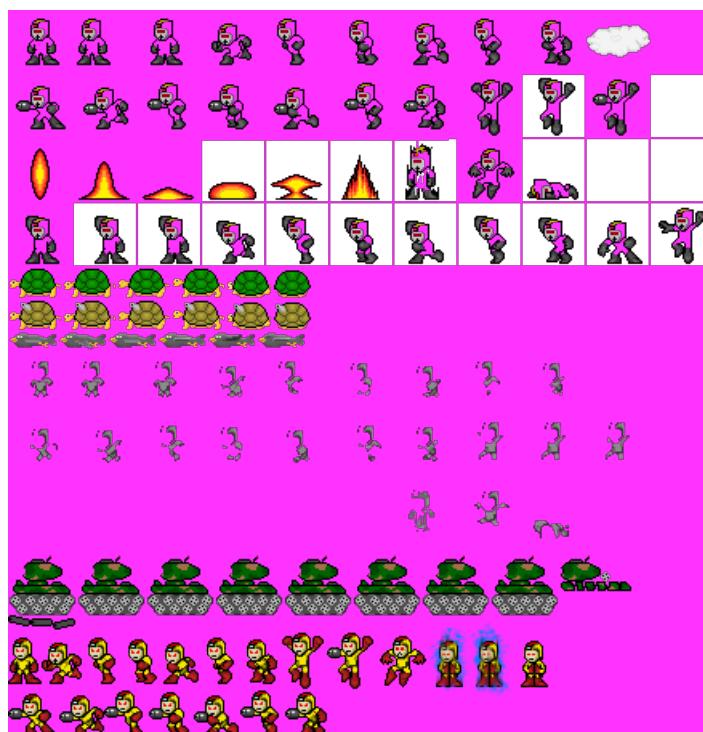


Abbildung 9.6: actors.png

Die obere Abbildung 9.6 zeigt den Texturatlas mit allen Spielfiguren. Die in Magenta dargestellten Flächen sind im Original transparent, die Hinterlegung erleichtert die Betrachtung. Durch die Aufteilung der Spielfigur in eine Außenumrandung (die oberen beiden Spritezeilen) und die Innenflächen (die beiden Spritezeilen unterhalb der Vögel) ist es möglich die Spielfigur softwareseitig einzufärben. Die Spielergrafiken basieren auf den Grafiken des d-man_work.png. Die Abbildung ist in Originalgröße und besitzt eine Breite von 440 Pixel zu 454 Pixel Höhe.



Abbildung 9.7: objects.png

Die obere Abbildung 9.7 zeigt den Texturatlasm mit allen Spielementen. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Zu den Spielementen gehören alle Projektils, sammelbaren Items, die Waffen-Items, die Sterne des Hintergrundes und die sammelbaren Münzen. Die Abbildung ist in vierfacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 67 Pixel zu 71 Pixel Höhe.



Abbildung 9.8: hud.png

Die obere Abbildung 9.8 zeigt den Texturatlasm mit allen Elementen für die Displays der Game-Pads. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Es werden die Energieleisten welche softwareseitig eingefärbt werden können, so wie die Spielziele abgebildet. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 210 Pixel zu 87 Pixel Höhe.

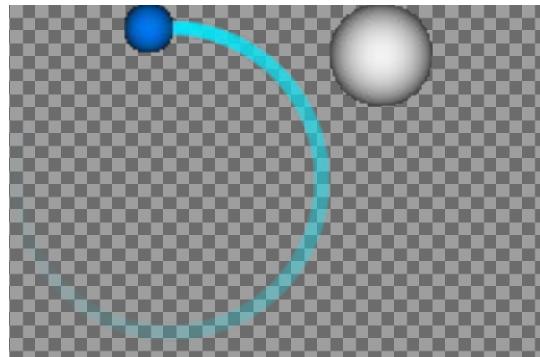


Abbildung 9.9: menu.png

Die obere Abbildung 9.9 zeigt den Texturatlasm mit den Animationsinhalten des Hauptmenüs. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Die graue Kugel wird softwareseitig eingefärbt um die Elemente des Nukleus dar zu stellen. Das blaue Elektron mit ins transparente verlaufendem Schweif wird für die Animation im Menü softwareseitig gedreht. Die Abbildung ist in Originalgröße dargestellt und besitzt eine Breite von 200 Pixel zu 132 Pixel Höhe.

Für das Design der Level wurden spezielle Tilesets (Zusammenstellung einzelner Grafikelemente welche zur Gestaltung verwendet werden können) eingesetzt. Diese können wie in Abschnitt 9.3.2 Leveldesign des Spiels beschrieben zu voll funktionsfähigen Levels zusammengesetzt werden und vom Spiel interpretiert werden. Die folgenden Abschnitte beschreiben die Grafiken und die Intention der verschiedenen Umgebungen.



Abbildung 9.10: tileset-oldstyle.png

Die obere Abbildung 9.10 zeigt das Tileset „oldstyle“. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 112 Pixel zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist dem Level das Aussehen einer alten Fabrik oder Kanalisation zu geben. Dieses Tileset ist direkt aus der Vorlage eines Mega Man 2 Tileset aus Quelle [I23] entstanden.

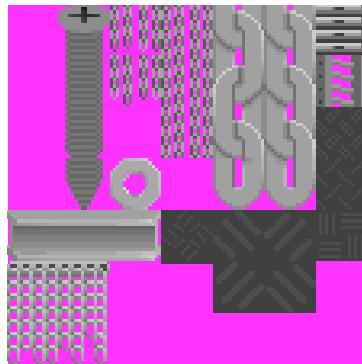


Abbildung 9.11: tileset-chain.png

Die obere Abbildung 9.11 zeigt das Tileset „chain“. Die mit Magenta hinterlegten Flächen sind im Original transparent. Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in doppelter Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 112 Pixel, zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist mit mehreren Sichtebenen im Level zu arbeiten. Ein Zusätzlicher Vordergrund und Hintergrund gibt dem Level mehr Dynamik. Als Darstellung wurden Stahlplatten, Ketten und Stahlträger gewählt.



Abbildung 9.12: tileset-forest.png

Die obere Abbildung 9.12 zeigt das Tileset „forest“. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 112 Pixel zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist dem Level das Aussehen von einer Waldumgebung oder Bäumen zu geben.

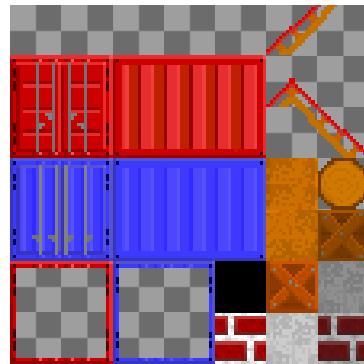


Abbildung 9.13: tileset-warehouse.png

Die obere Abbildung 9.13 zeigt das Tileset „warehouse“. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 112 Pixel zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist dem Level das Aussehen einer Lagerhalle zu geben. Auch hier wurde wieder mit verschiedenen Sichtebenen gearbeitet.

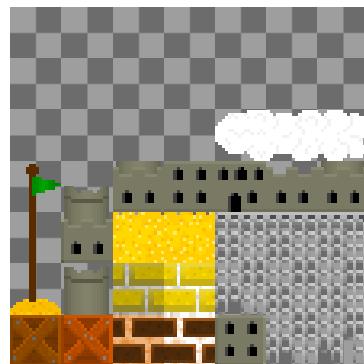


Abbildung 9.14: tileset-desert.png

Die obere Abbildung 9.14 zeigt das Tileset „desert“. Die mit grauen Quadraten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 112 Pixel, zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist dem Level das Aussehen einer Wüstenlandschaft zu geben. Als Inspiration diente Mario Bros (1986) auf dem Nintendo Entertainment System [I24].



Abbildung 9.15: tileset-ship.png

Die obere Abbildung 9.15 zeigt das Tileset „ship“. Die mit grauen Quadranten hinterlegten Flächen sind im Original transparent. Jedes Quadrat besitzt eine Größe von 8x8 Pixeln (Breite x Höhe). Jedes Tile ist 16x16 Pixel (Breite x Höhe) groß. Diese Grafikbausteine können verwendet werden um die Ebenen der Level zu gestalten. Die Abbildung ist in zweifacher Vergrößerung dargestellt um die Elemente besser erkennen zu können und besitzt eine Breite von 224 Pixel zu 112 Pixel Höhe. Die Idee hinter dieser Gestaltung ist dem Level ein maritimes Aussehen zu geben. Aus den Tiles können zum Beispiel Schiffe oder Hafenlandschaften erstellt werden.

9.3.2 Leveldesign des Spiels

Die Levels wurden mit Tiled[©] erstellt. Dieses Programm ermöglicht es mittels grafischen Werkzeugen eine XML Struktur zu erstellen. Informationen zu Tiled[©] können [I25] entnommen werden.

Die Level wurden mit drei sichtbaren Ebenen und einer Objektebene konzipiert. Dies ermöglicht die Gestaltung von Vorder- und Hintergrund, einer Spielebene, so wie das setzen von Spawnpunkten auf denen Gegner, Spieler und Objekte entstehen können. Für Parallax-scrollen oder ähnliche Tiefeneffekte werden weitere Ebenen benötigt.

Die Tileebenen für den Vorder- und Hintergrund sind rein dekorativer Natur, die Tileebene Zwischen diesen beiden Ebenen wurde bereits als Spielebene bezeichnet. Diese Ebene stellt sich als solide für den Spieler dar. Alle Wege, Decken, Böden und Hindernisse gehören somit zur Spielebene und können nicht von Waffen, Spielern und den meisten Gegnern durchquert werden. Bei der Erstellung der Spielebene ist darauf zu achten, welche Bereiche von den Spielfiguren betreten werden können und welche unzugänglich für den Spieler sein sollen.

Die Objektebene steht außerhalb der drei Tileebenen. Objekte die auf dieser Ebene platziert werden, müssen vom Programm so interpretiert werden, als wären sie auf der Spielebene. Die Spawnpunkte für Gegner (Rote Quadrate), Objekte (Waffen: gelbe Quadrate, Munition und Lebensenergie: blaue Quadrate) und Spieler (grüne Quadrate) dienen lediglich als Markierungen, die auf ihnen erzeugten Dinge werden als Objekte auf die Spielebene eingefügt.

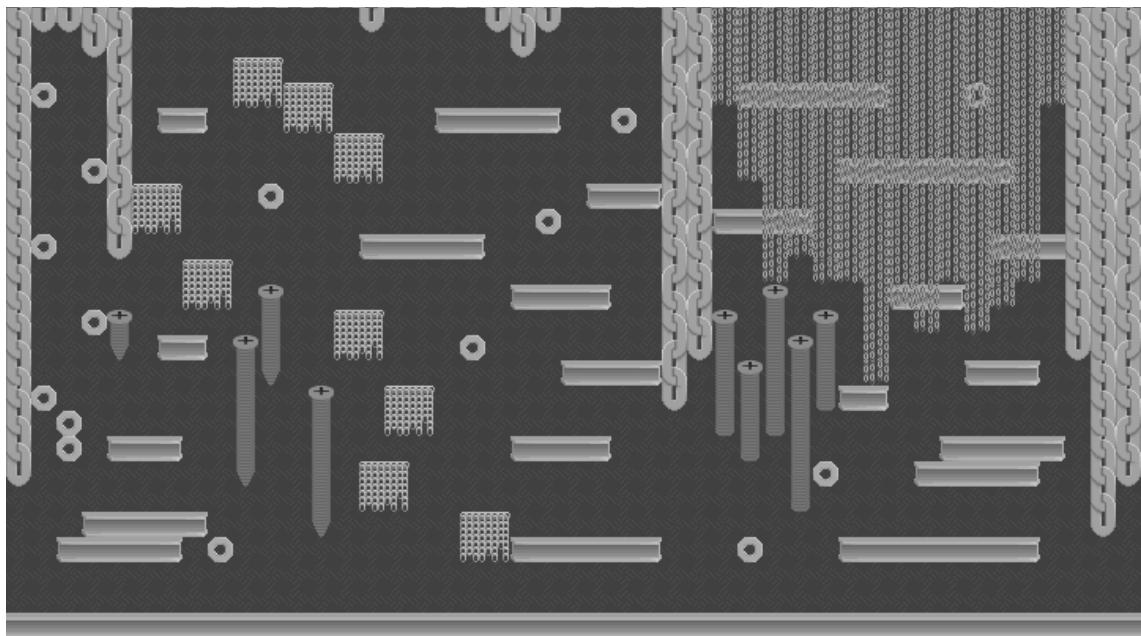


Abbildung 9.16: Übersicht aller Ebenen

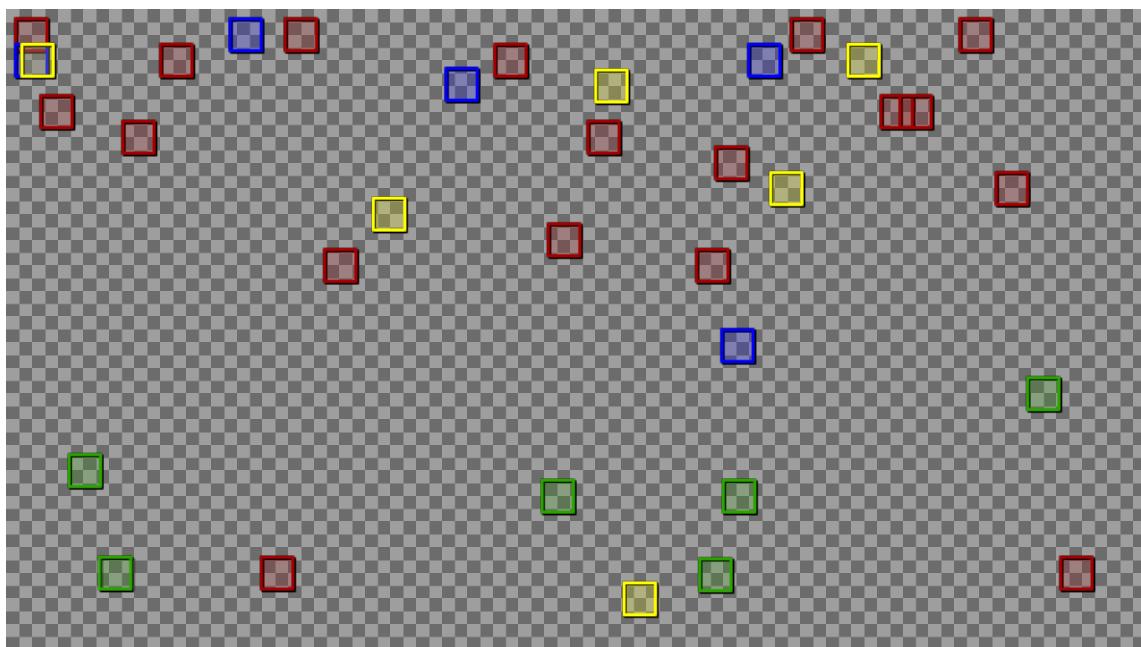


Abbildung 9.17: Objektebene

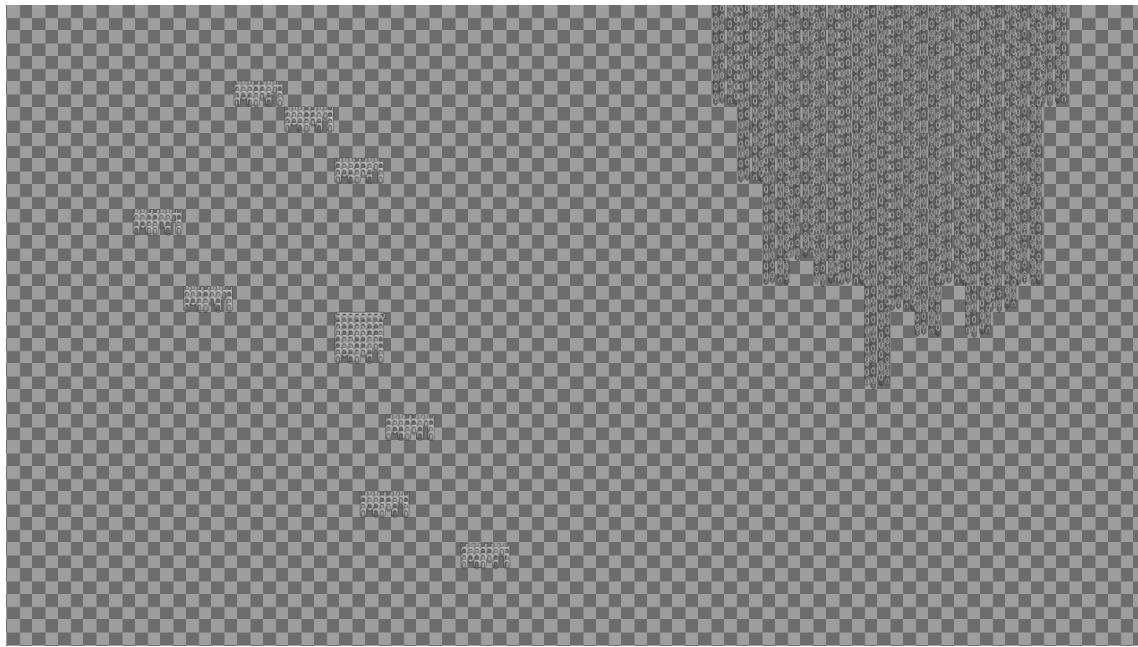


Abbildung 9.18: Vordergrund

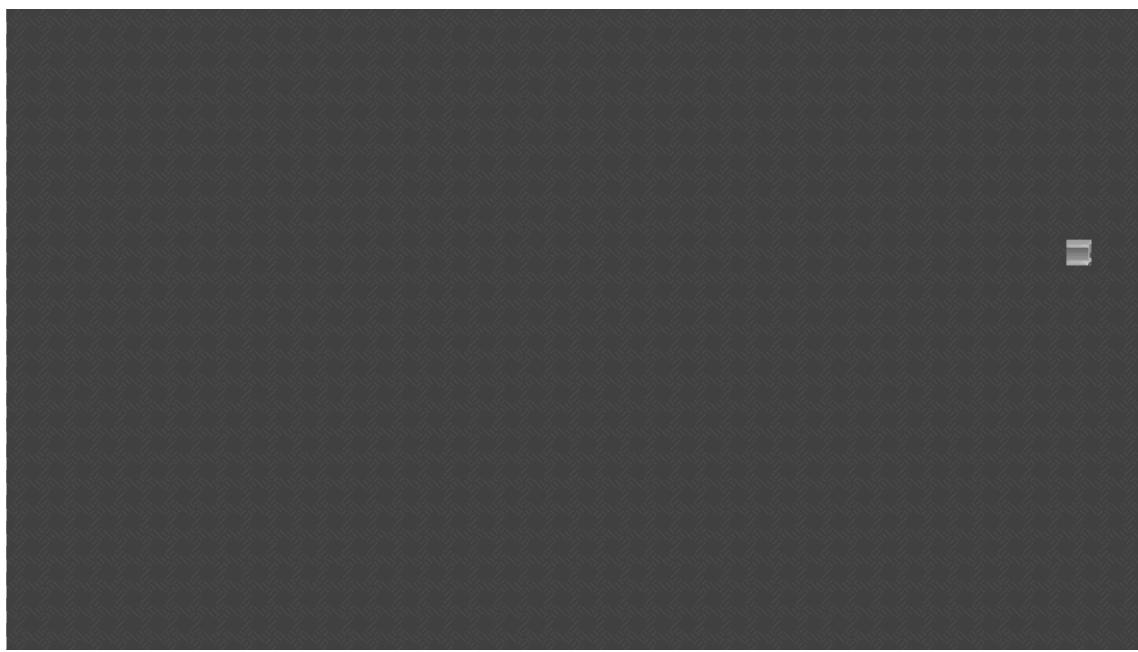


Abbildung 9.19: Hintergrund

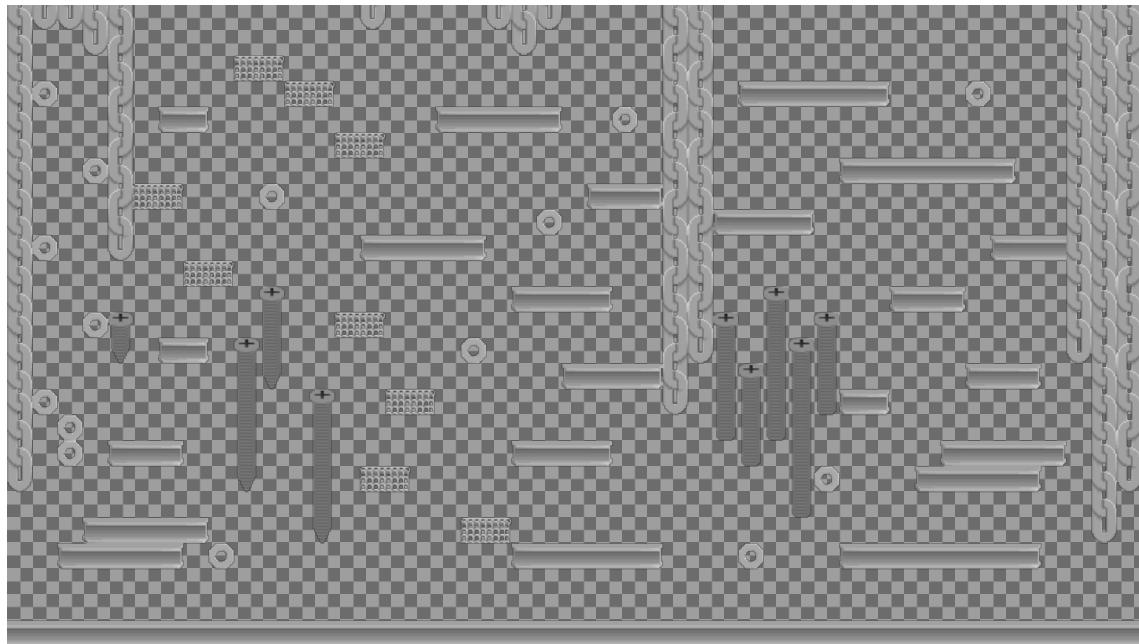


Abbildung 9.20: Hauptebene

Die obere Abbildung 9.16 zeigt eine Übersicht des Levels „Chains“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.17 der Objektebene, Abbildung 9.18 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.19 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.20 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

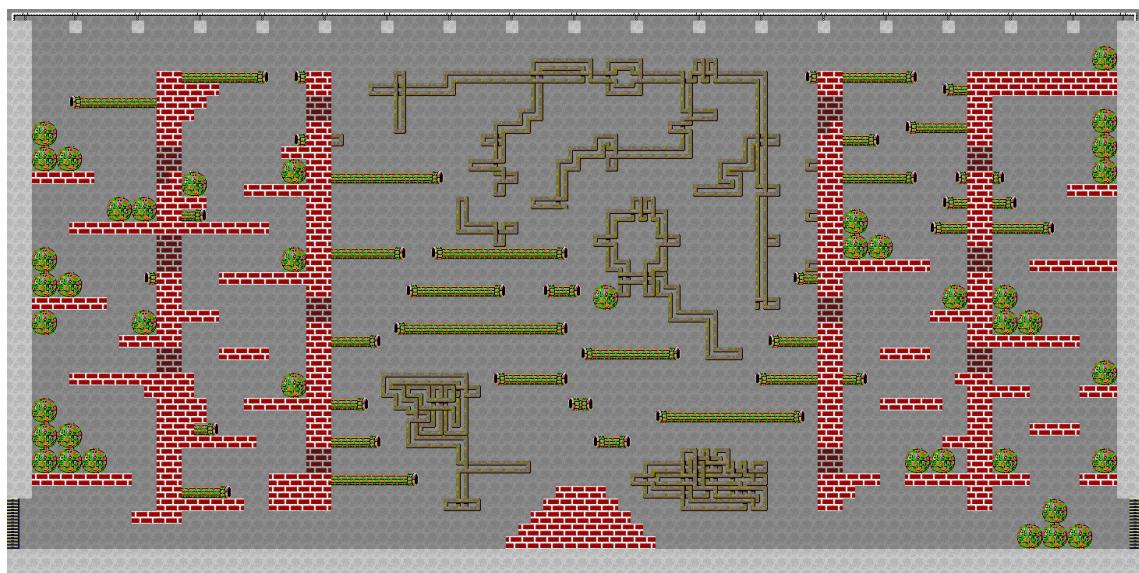


Abbildung 9.21: Übersicht aller Ebenen

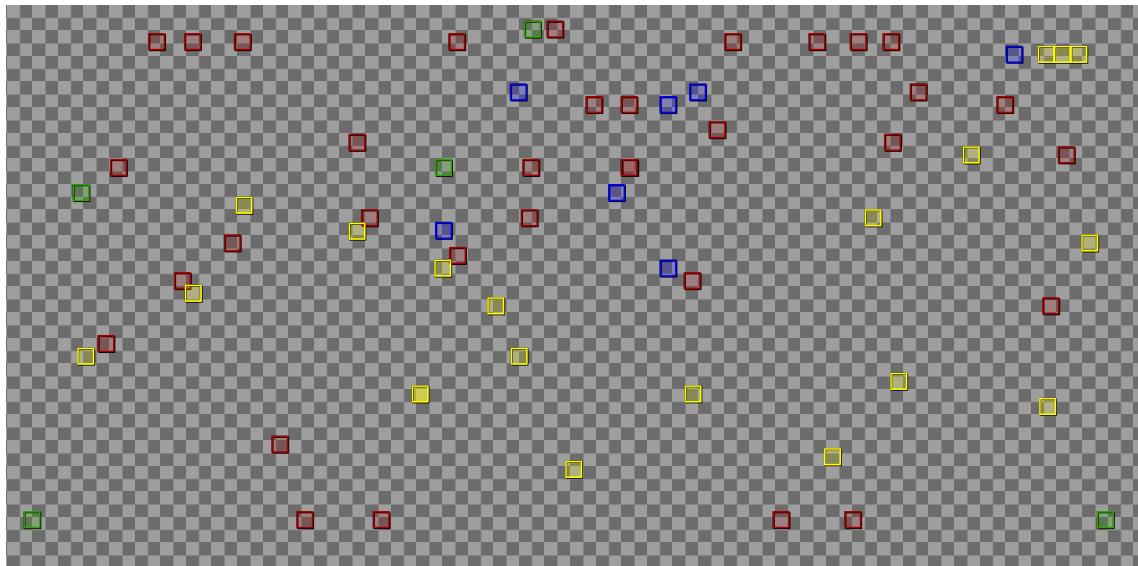


Abbildung 9.22: Objektebene

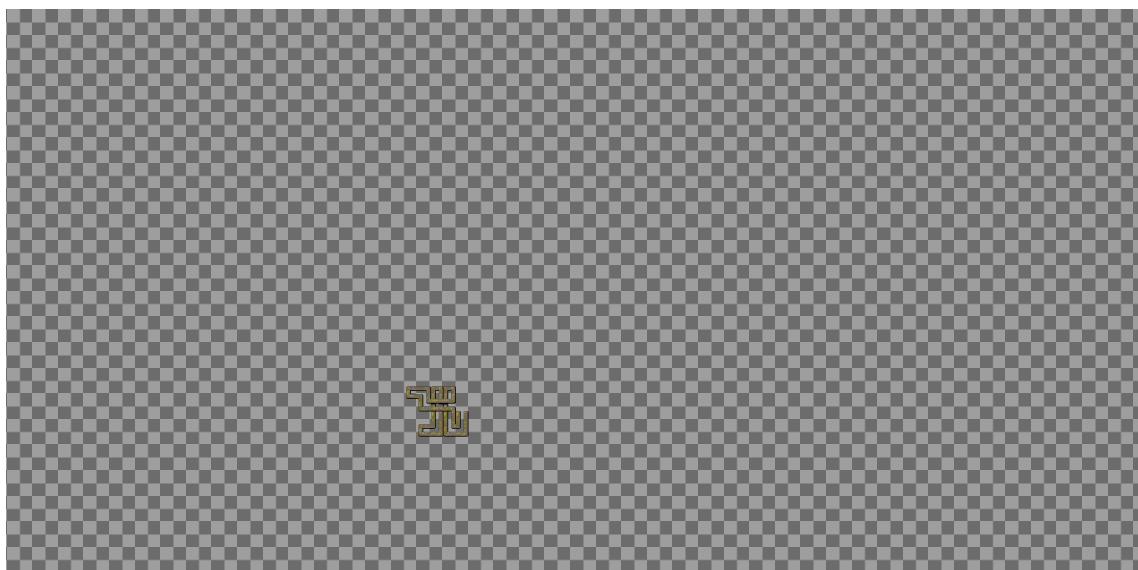
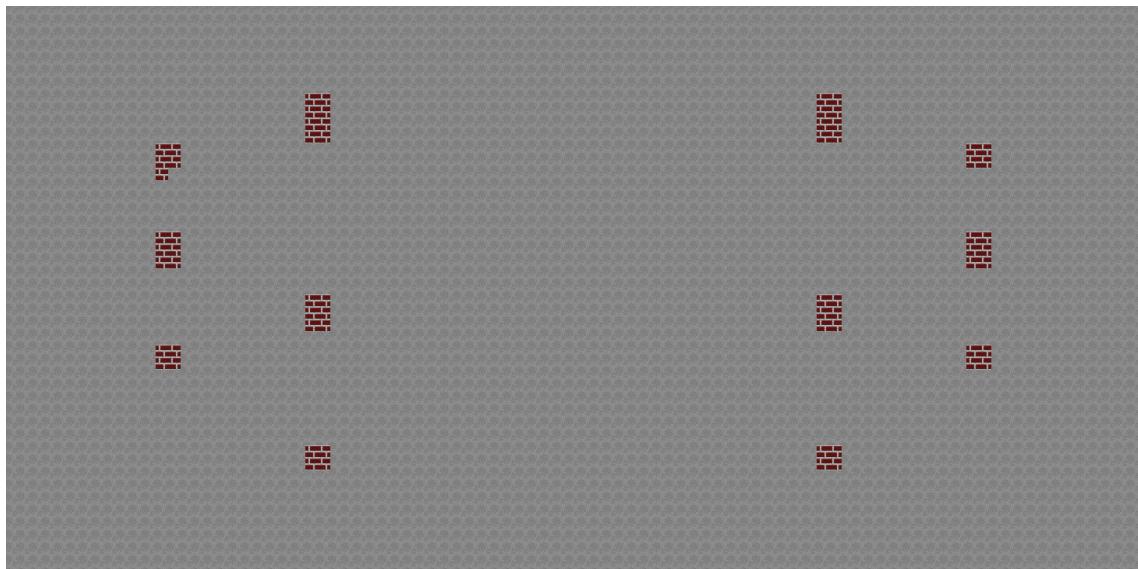
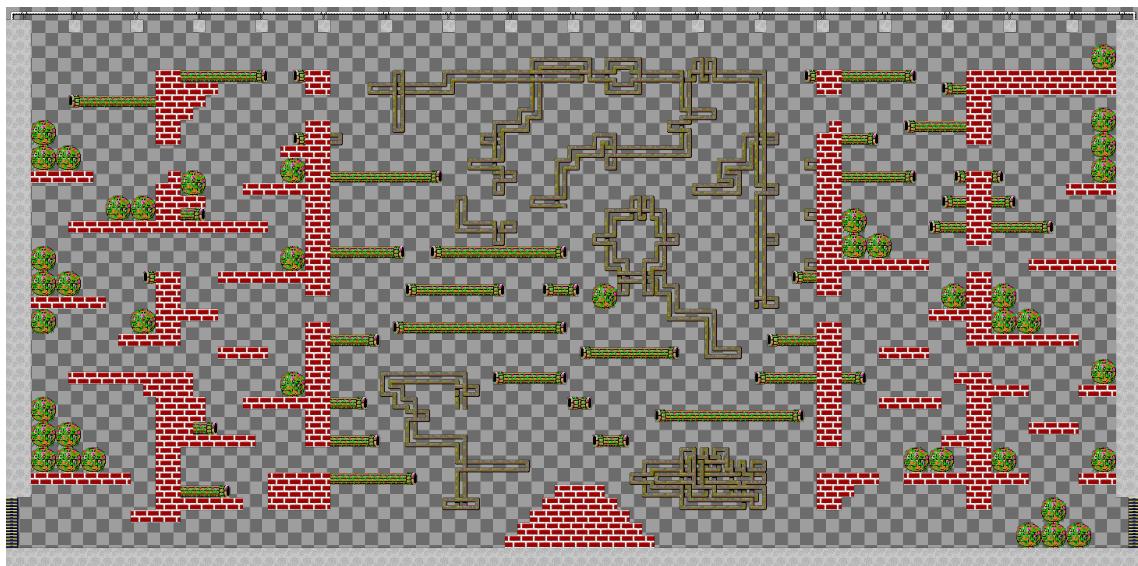


Abbildung 9.23: Vordergrund

**Abbildung 9.24: Hintergrund****Abbildung 9.25: Hauptebene**

Die obere Abbildung 9.21 zeigt eine Übersicht des Levels „OldFactory“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.22 der Objektebene, Abbildung 9.23 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.24 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.25 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

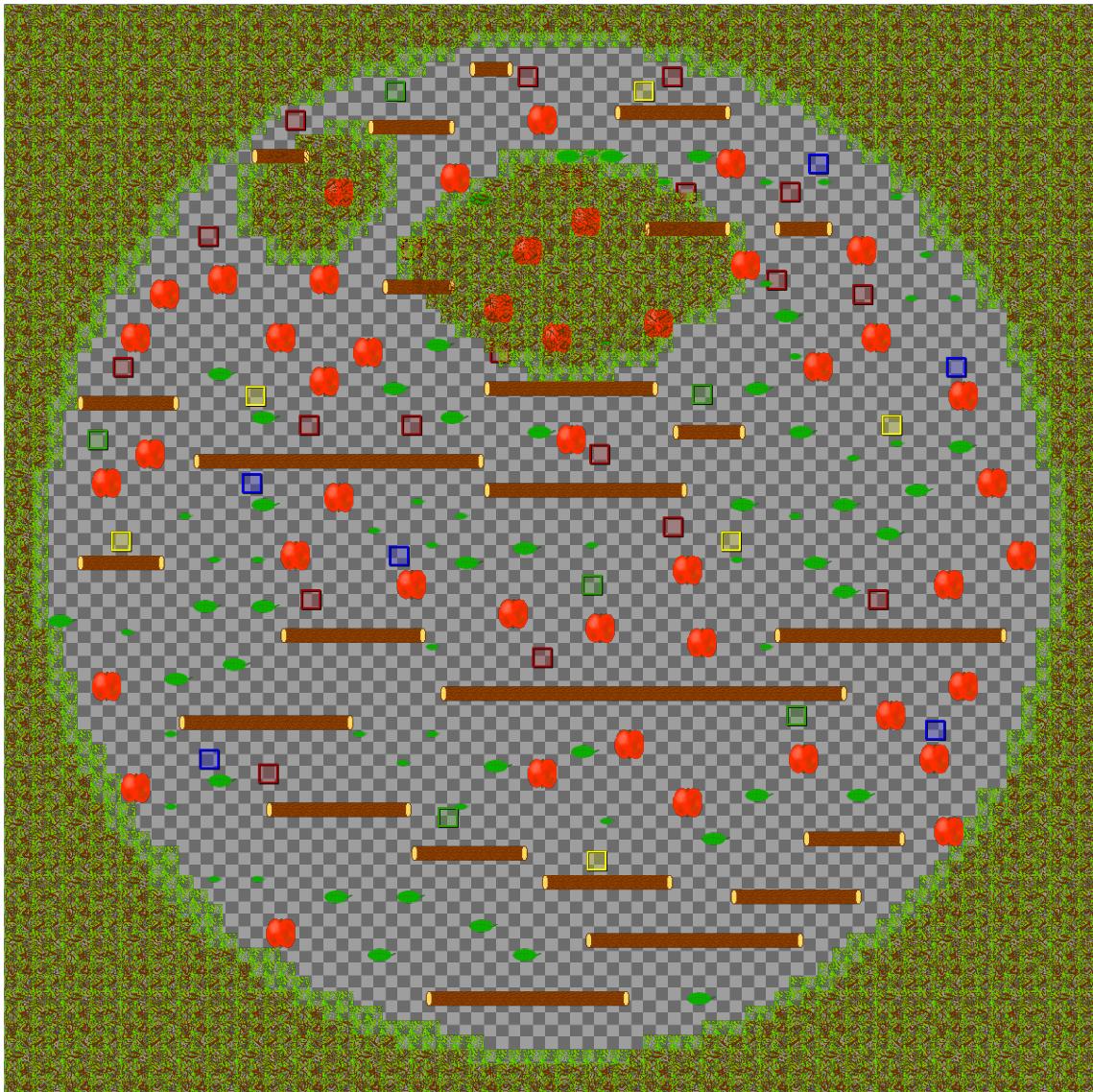


Abbildung 9.26: Übersicht aller Ebenen

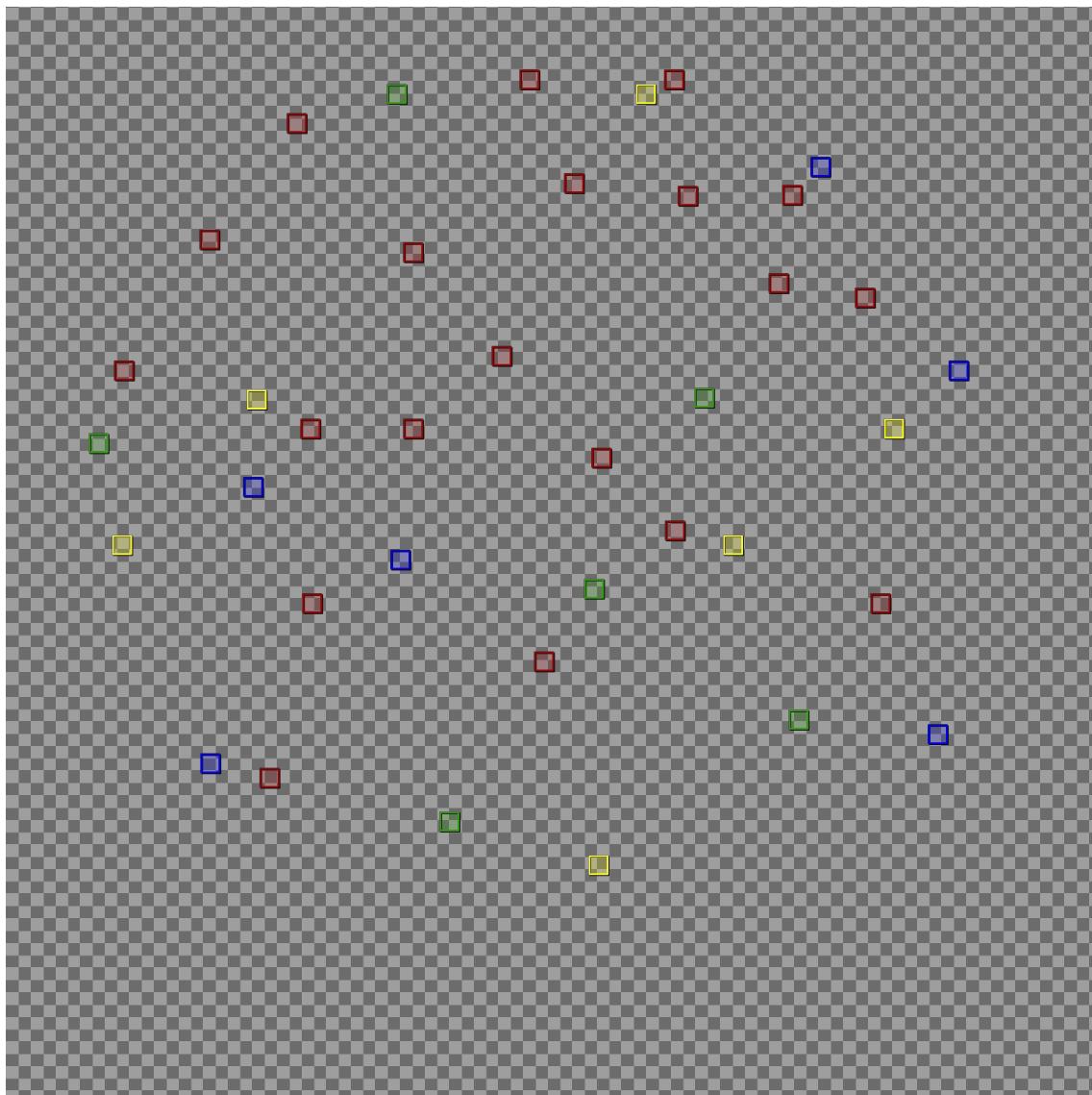


Abbildung 9.27: Objektebene

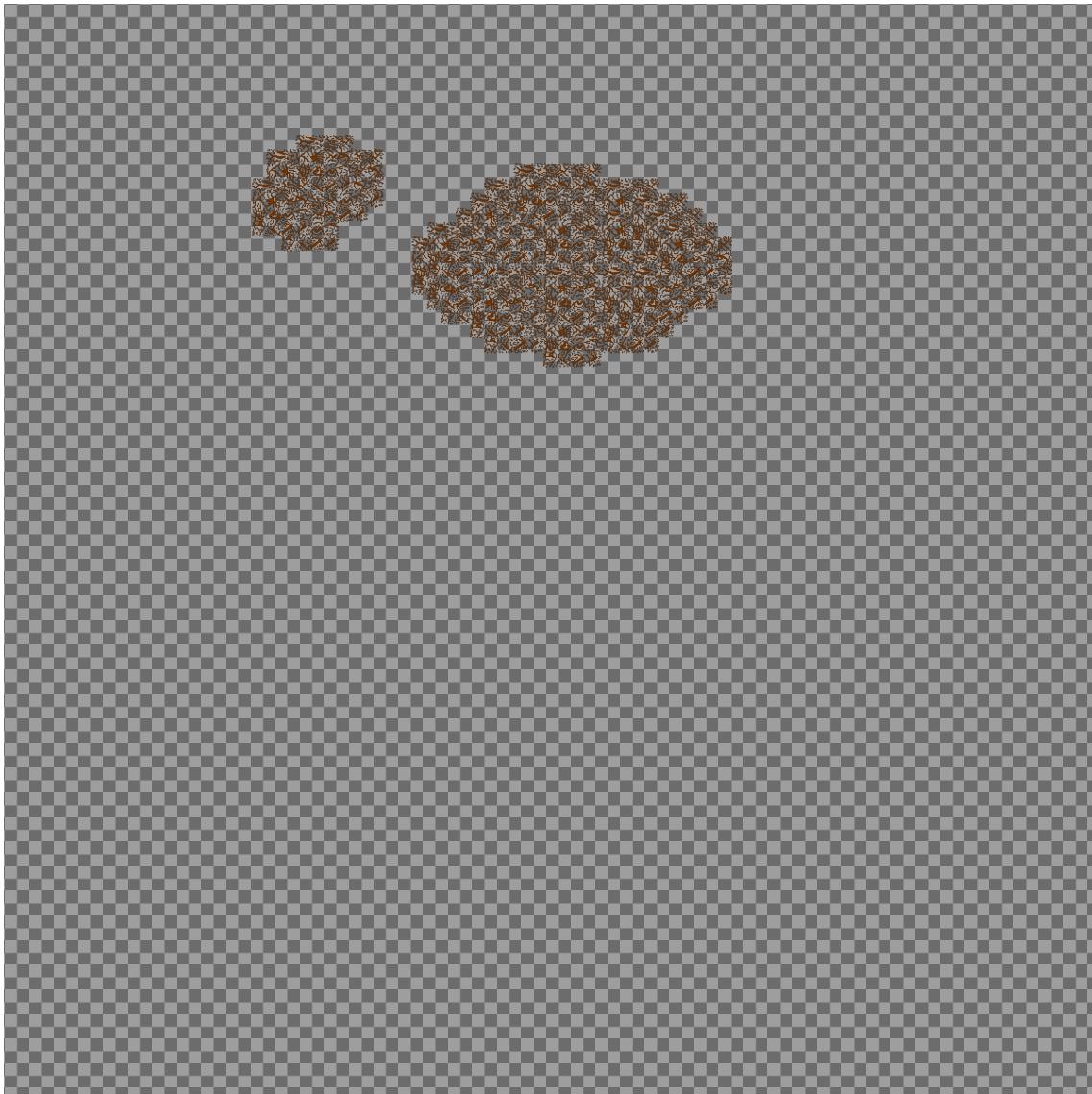


Abbildung 9.28: Vordergrund

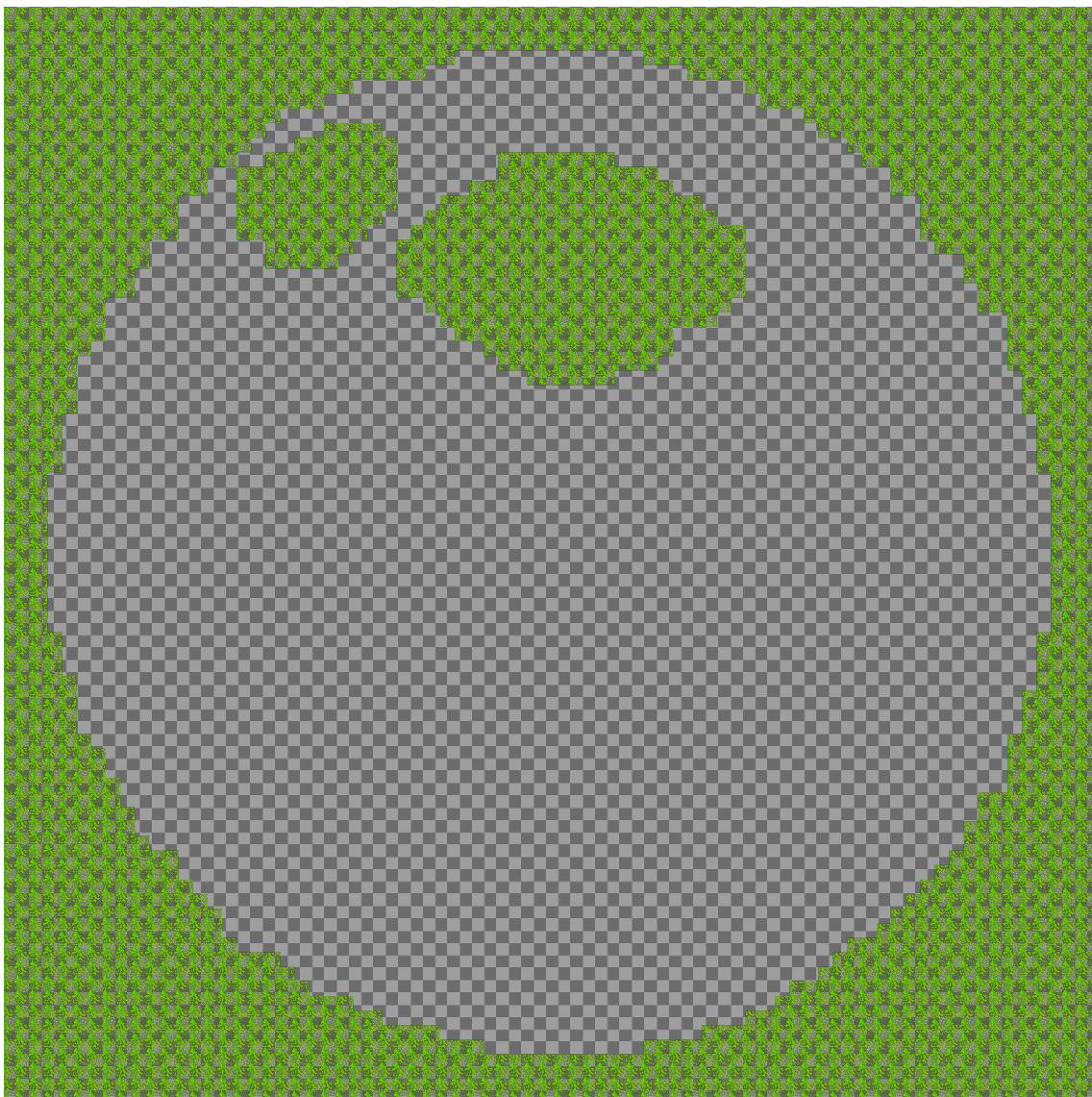


Abbildung 9.29: Hintergrund

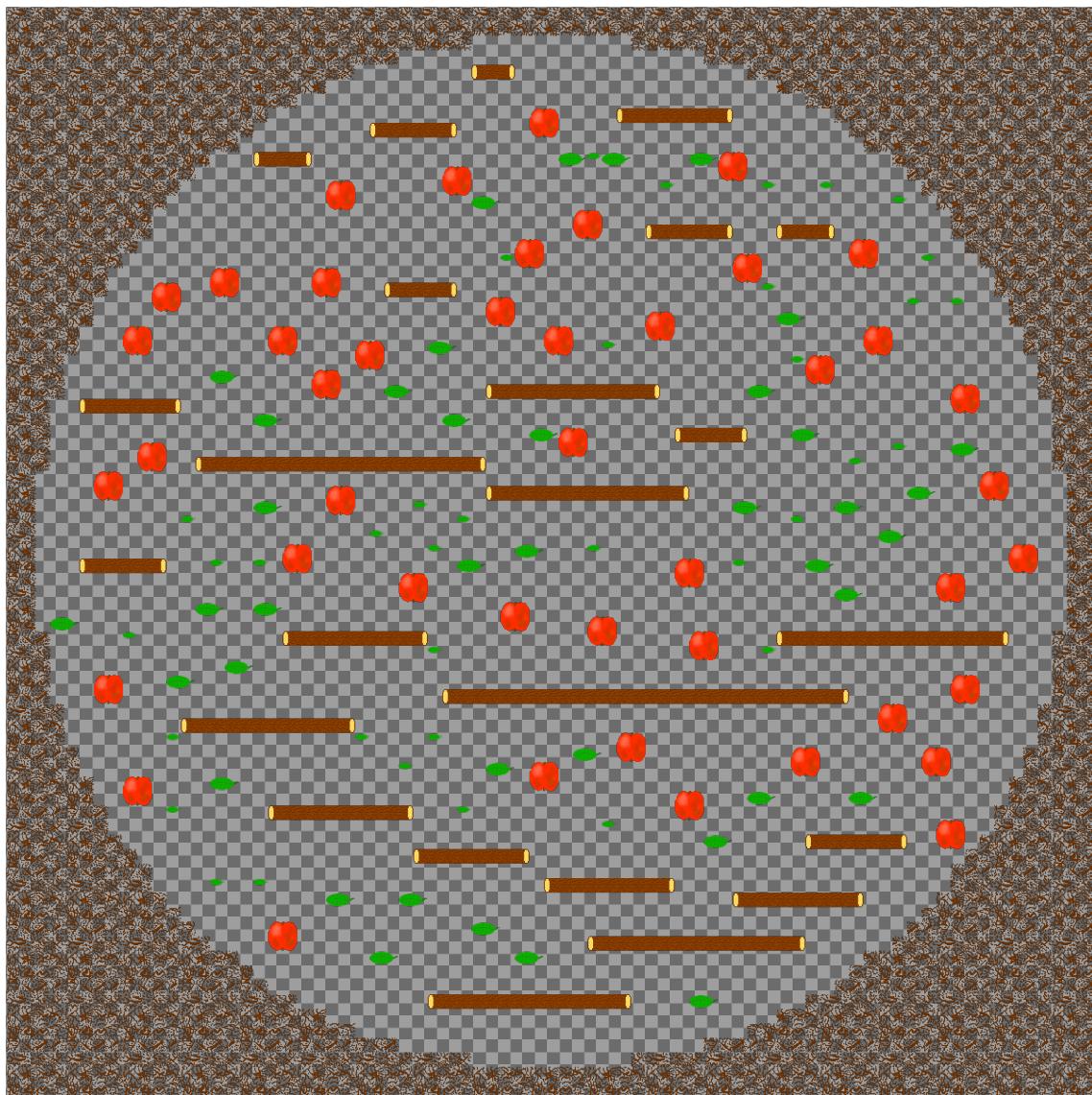


Abbildung 9.30: Hauptebene

Die obere Abbildung 9.26 zeigt eine Übersicht des Levels „Tree“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.27 der Objektebene, Abbildung 9.28 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.29 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.30 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.



Abbildung 9.31: Übersicht aller Ebenen

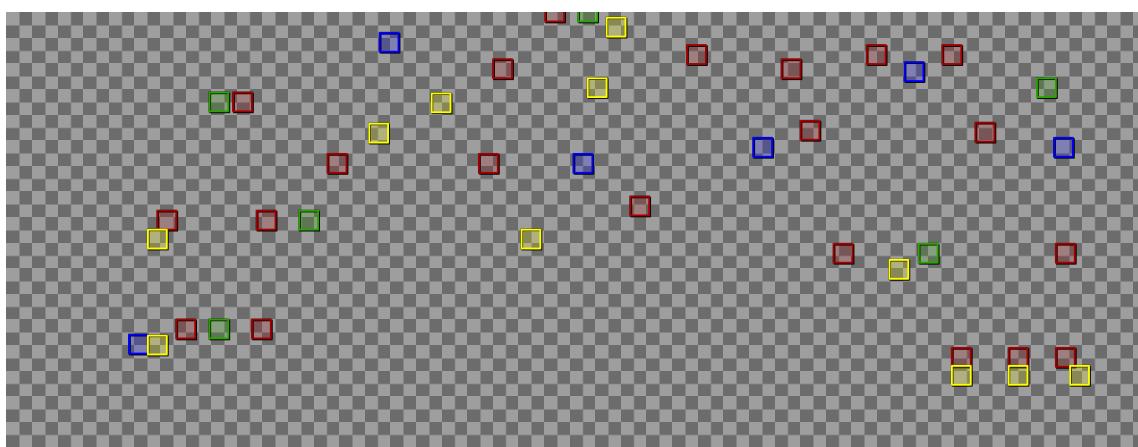


Abbildung 9.32: Objektebene

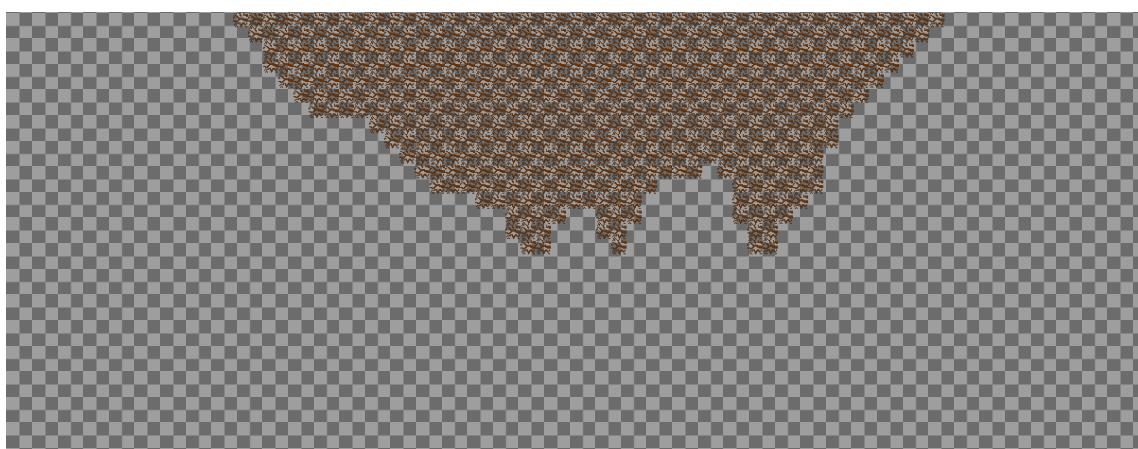


Abbildung 9.33: Vordergrund

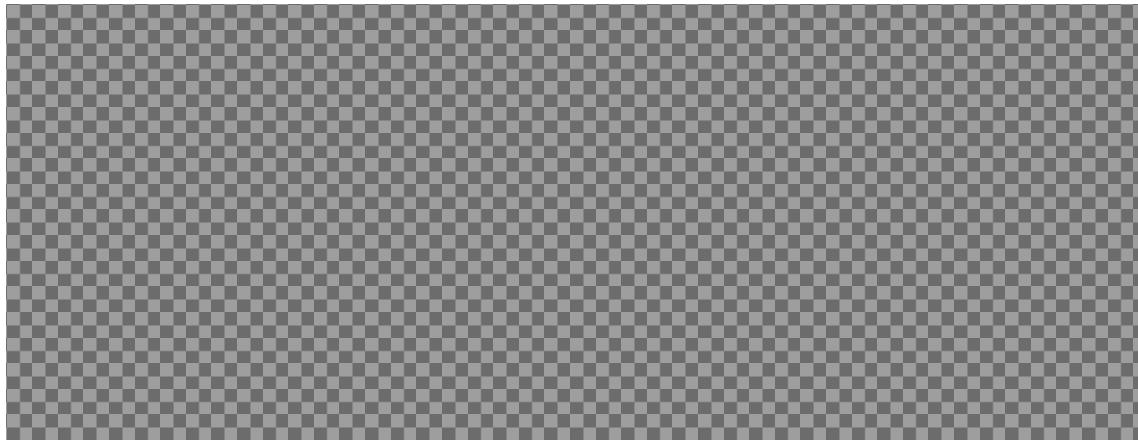


Abbildung 9.34: Hintergrund

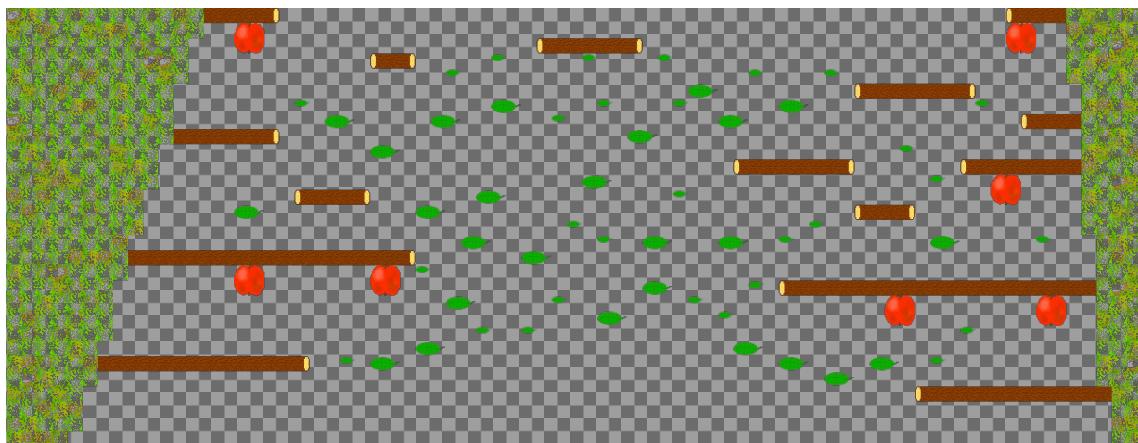


Abbildung 9.35: Hauptebene

Die obere Abbildung 9.31 zeigt eine Übersicht des Levels „TwoTrees“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.32 der Objektebene, Abbildung 9.33 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.34 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.35 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

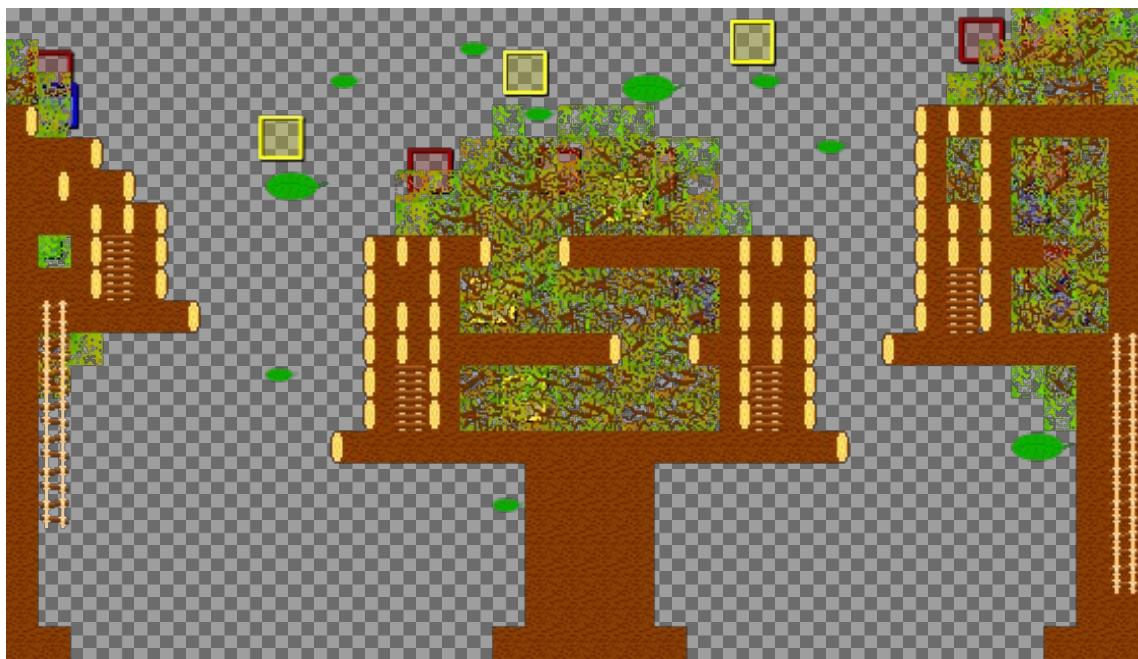


Abbildung 9.36: Übersicht aller Ebenen

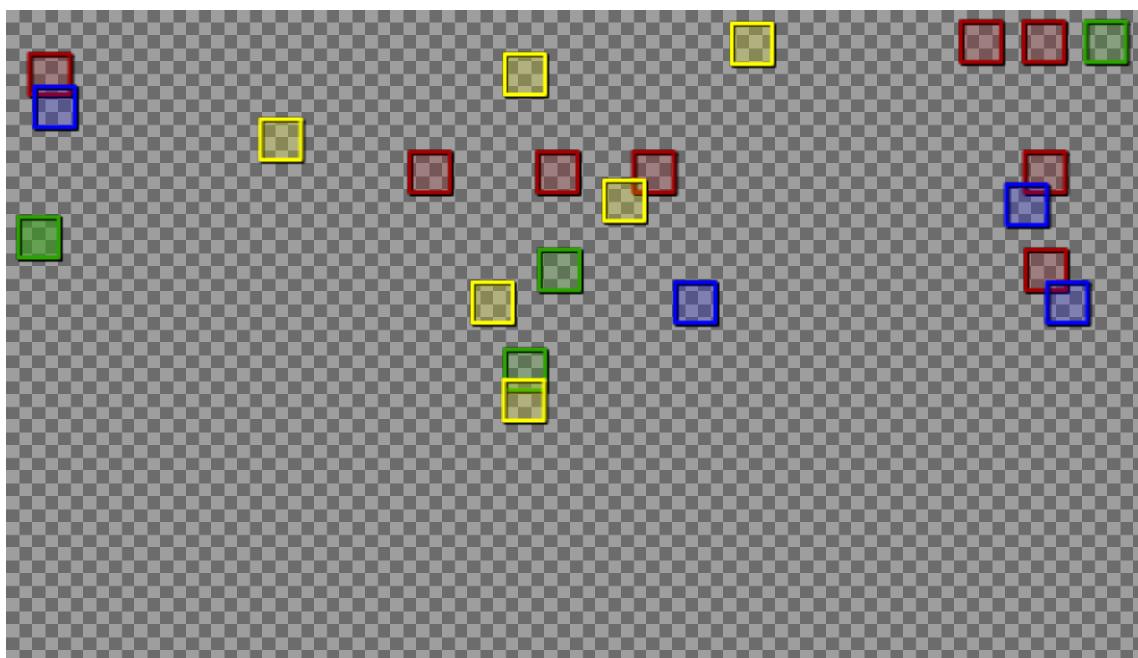


Abbildung 9.37: Objektebene

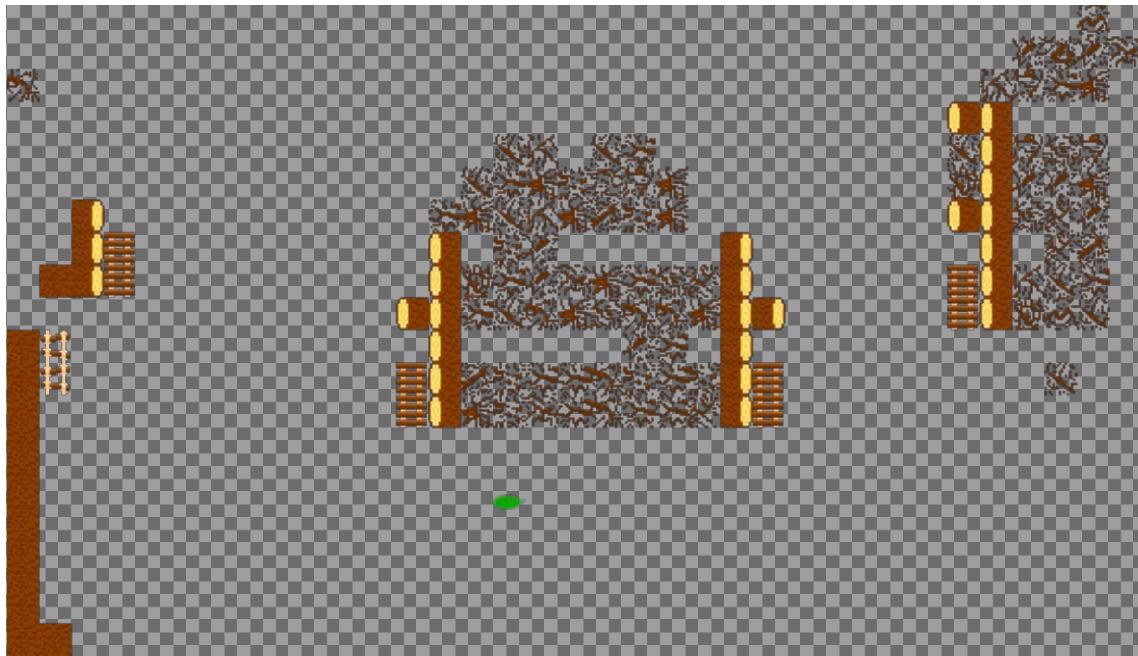


Abbildung 9.38: Vordergrund

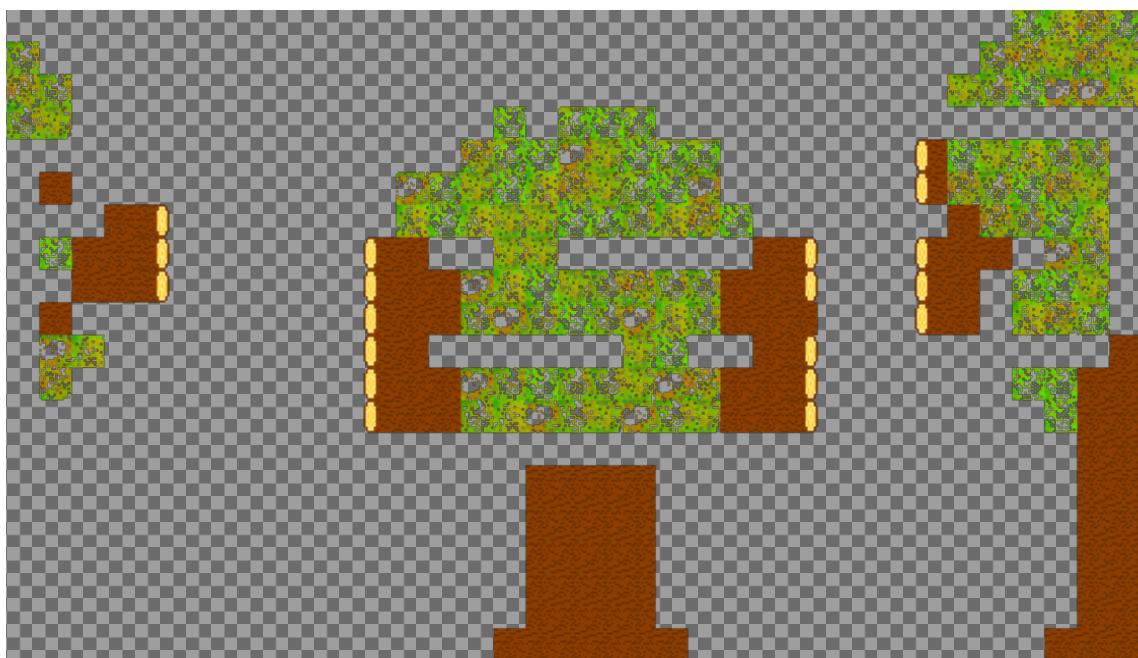


Abbildung 9.39: Hintergrund

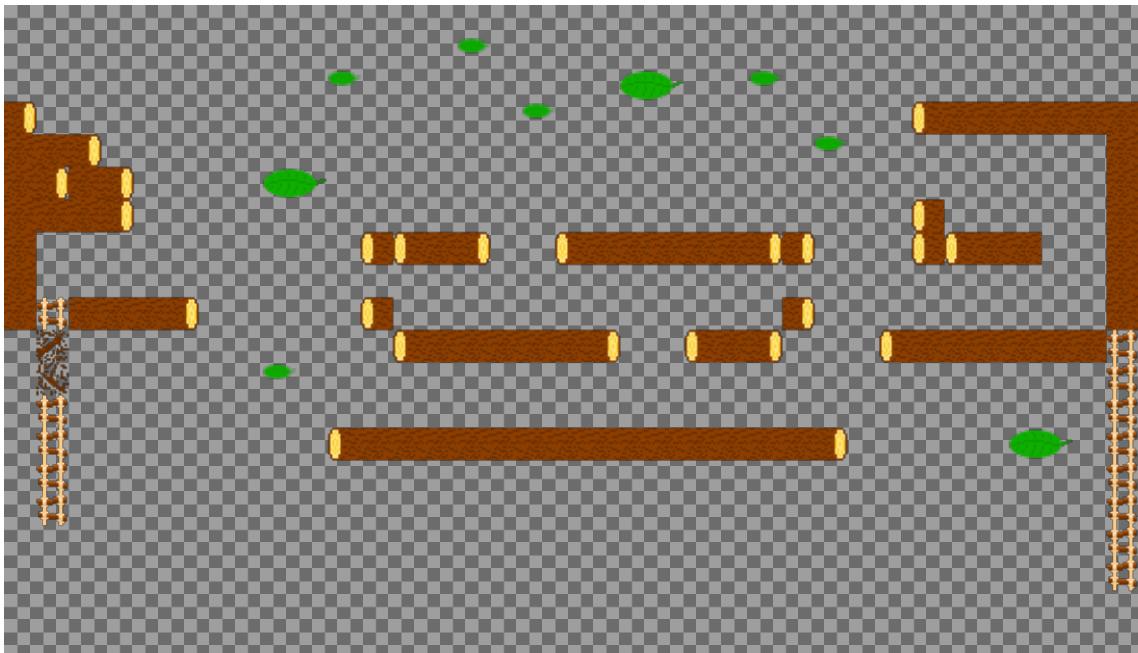


Abbildung 9.40: Hauptebene

Die obere Abbildung 9.36 zeigt eine Übersicht des Levels „Treehouse“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.37 der Objektebene, Abbildung 9.38 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.39 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.40 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

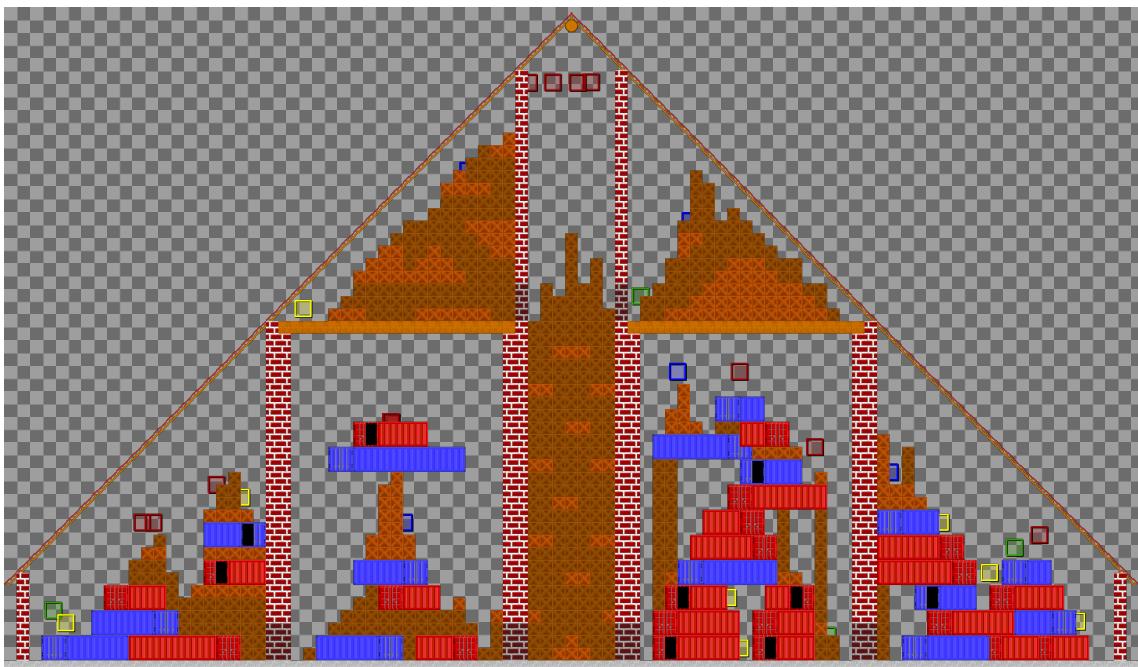


Abbildung 9.41: Übersicht aller Ebenen

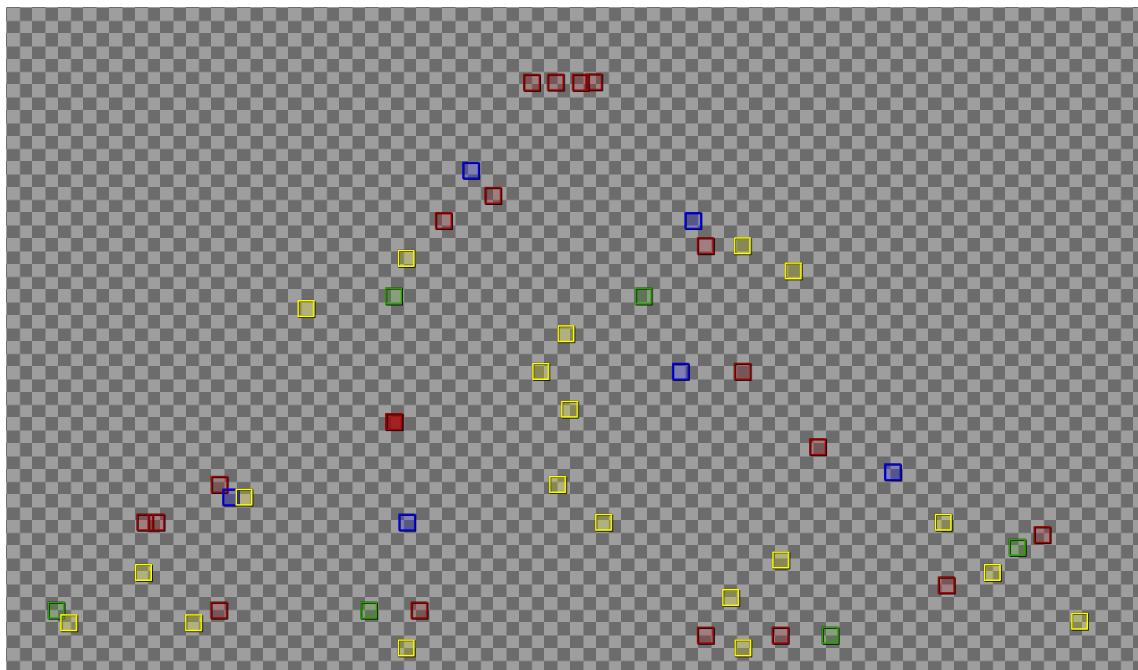


Abbildung 9.42: Objektebene

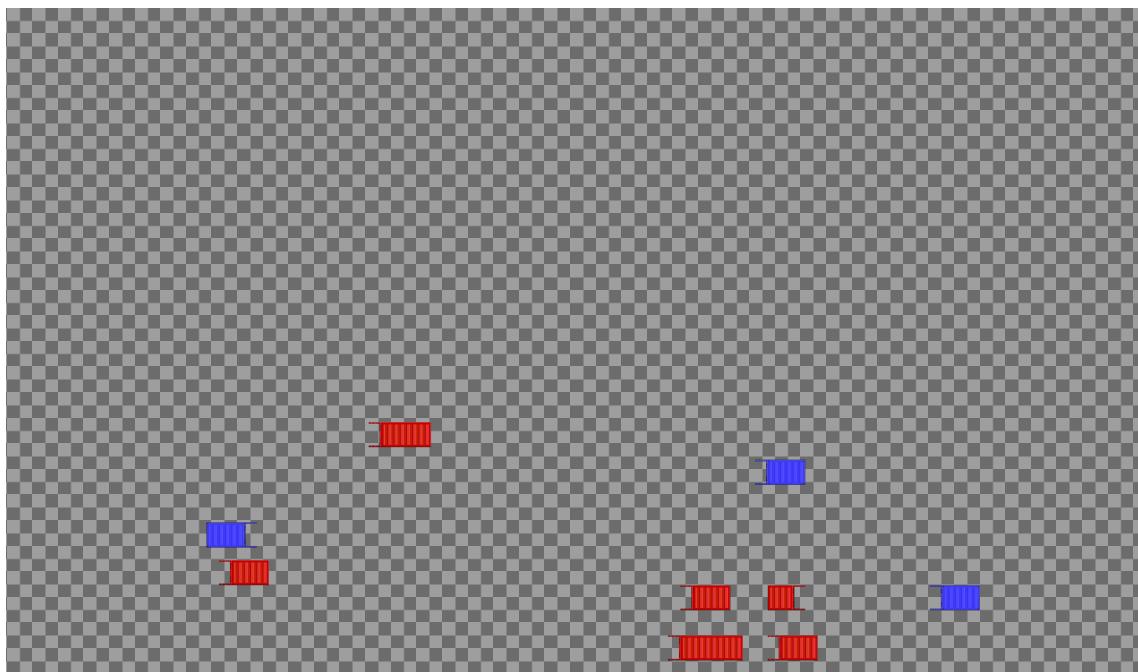


Abbildung 9.43: Vordergrund

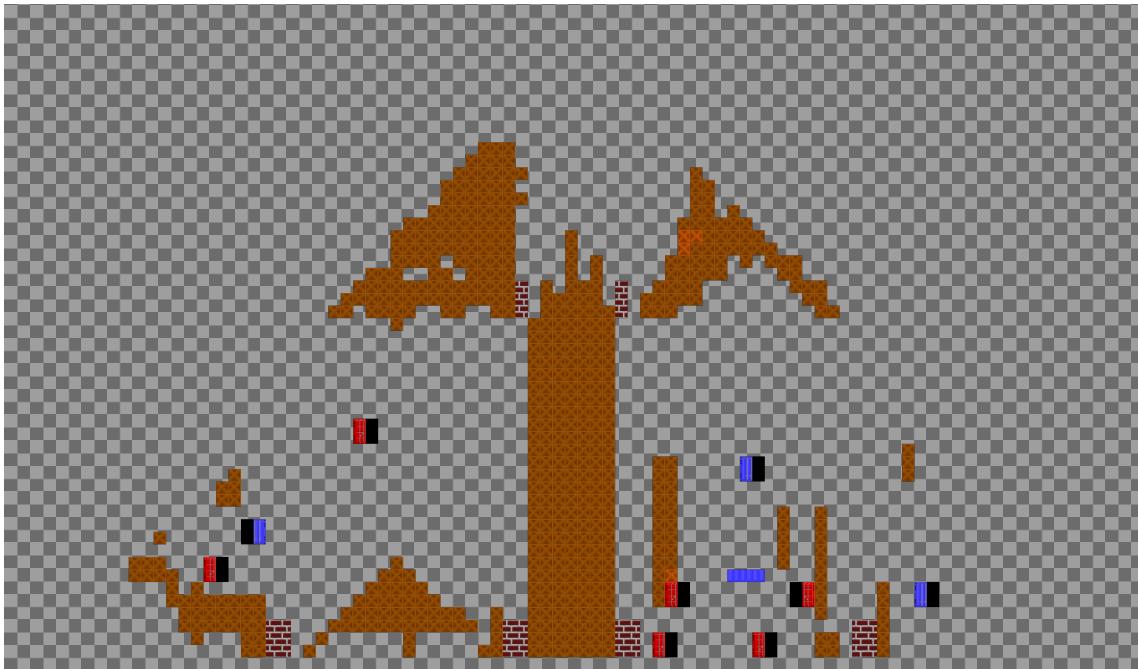


Abbildung 9.44: Hintergrund

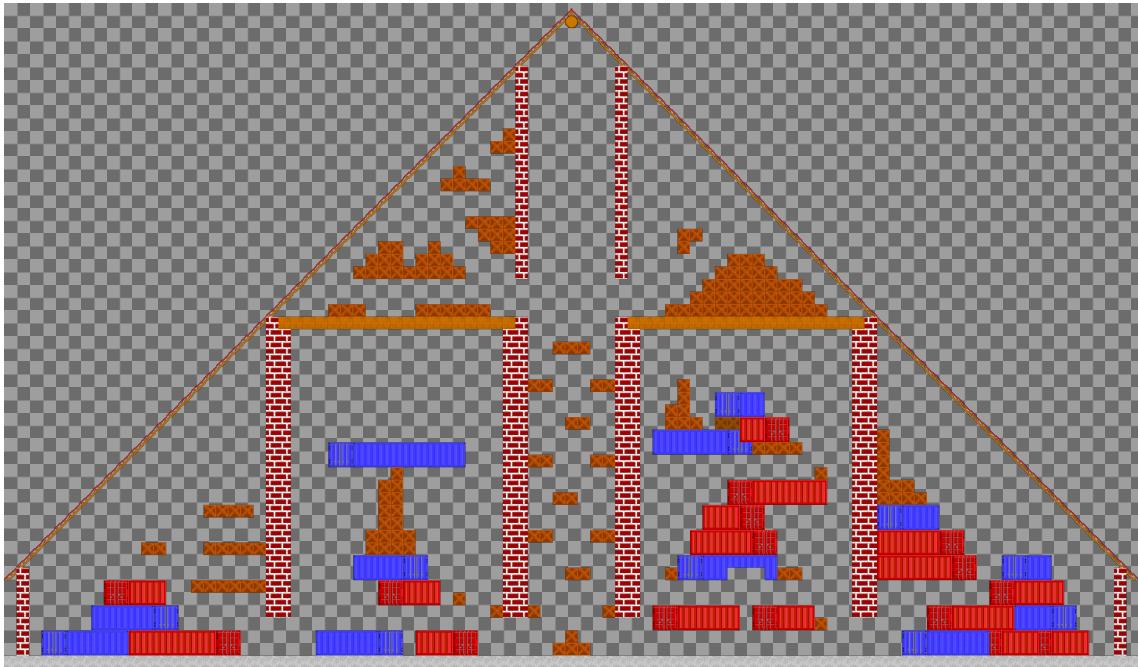


Abbildung 9.45: Hauptebene

Die obere Abbildung 9.41 zeigt eine Übersicht des Levels „Warehouse“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.42 der Objektebene, Abbildung 9.43 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.44 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.45 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

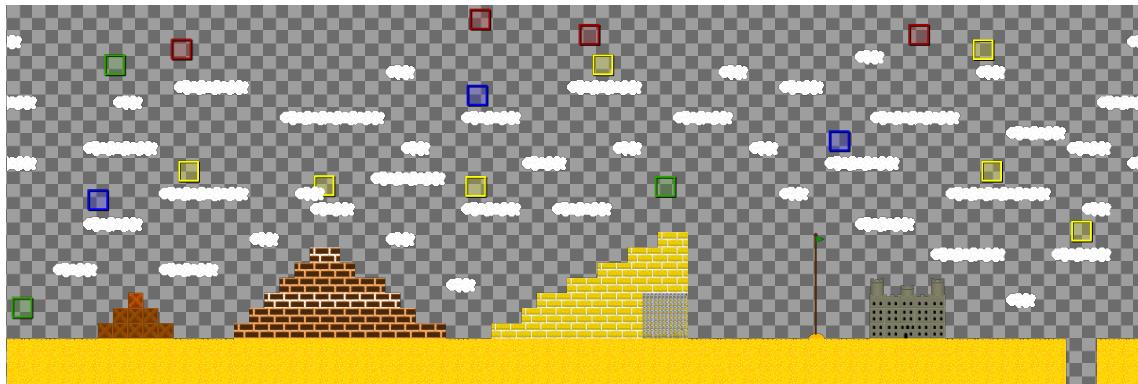


Abbildung 9.46: Übersicht aller Ebenen

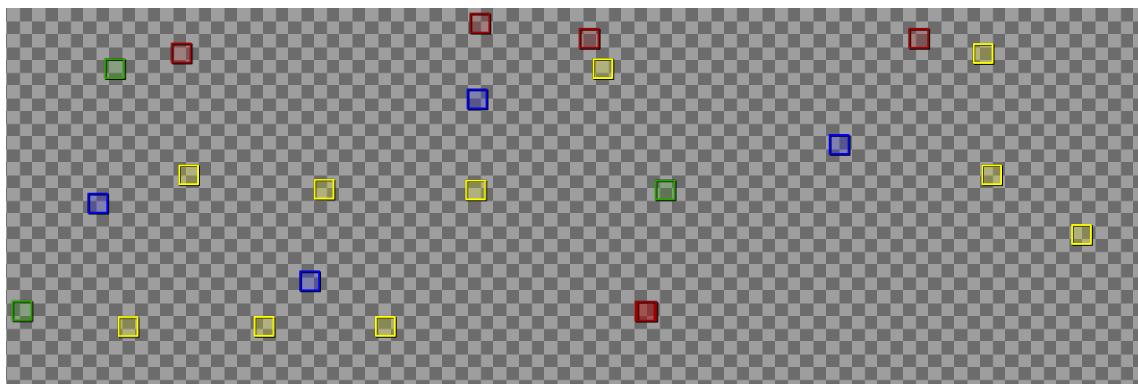


Abbildung 9.47: Objektebene

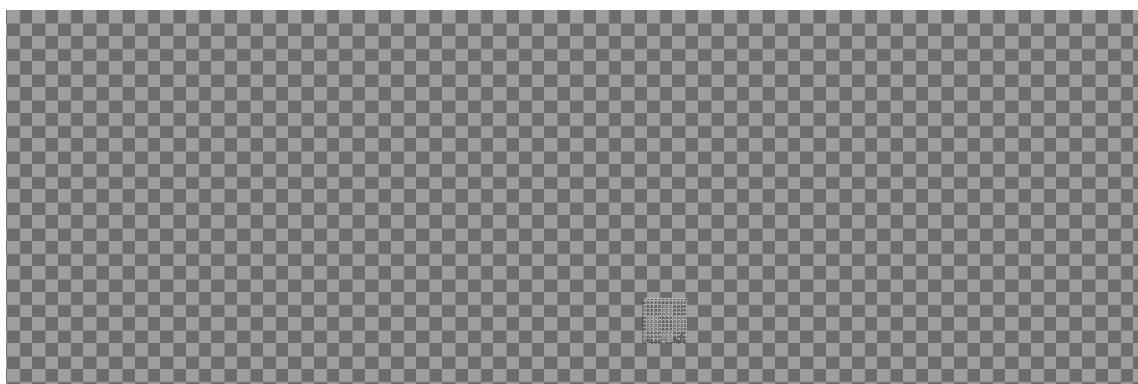


Abbildung 9.48: Vordergrund

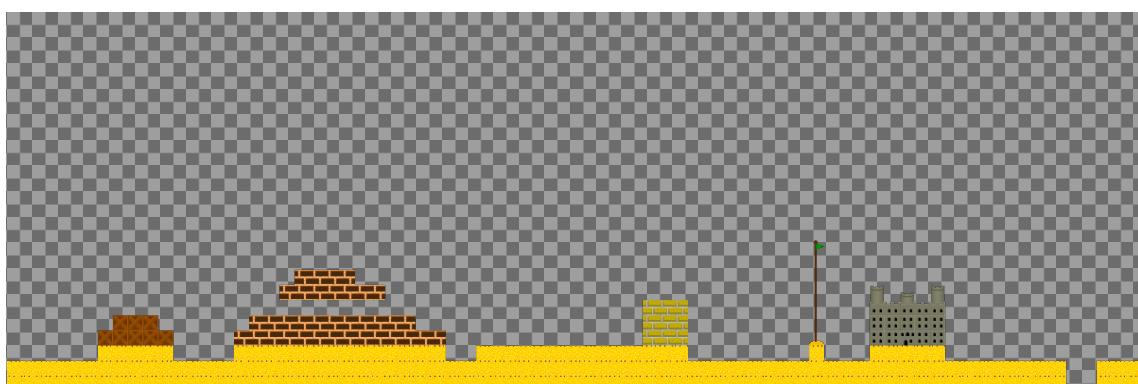
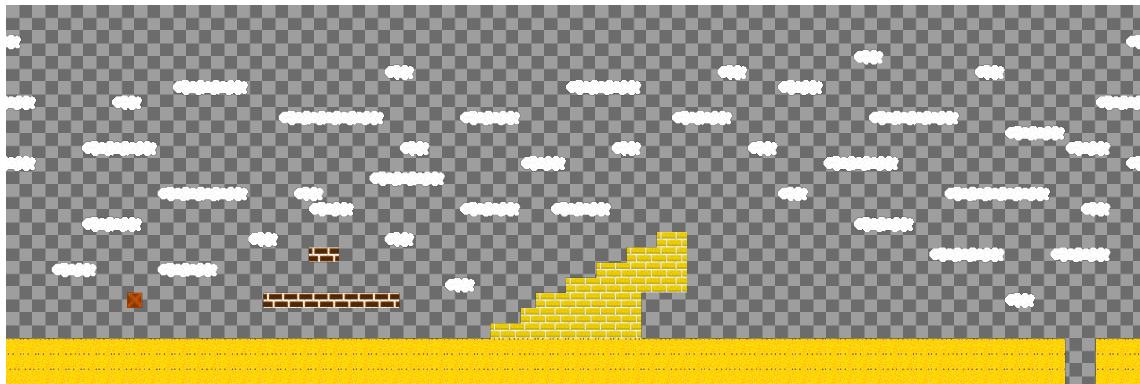
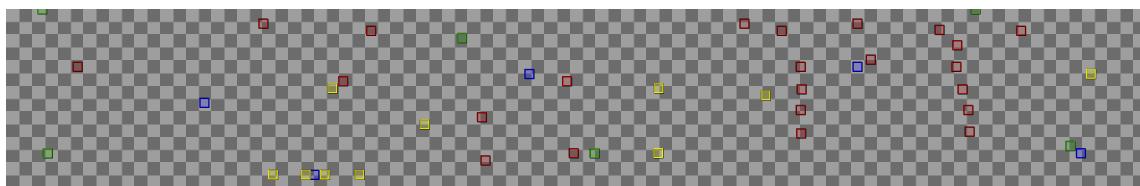
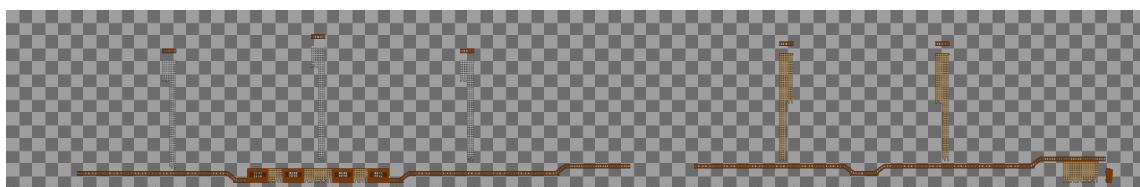
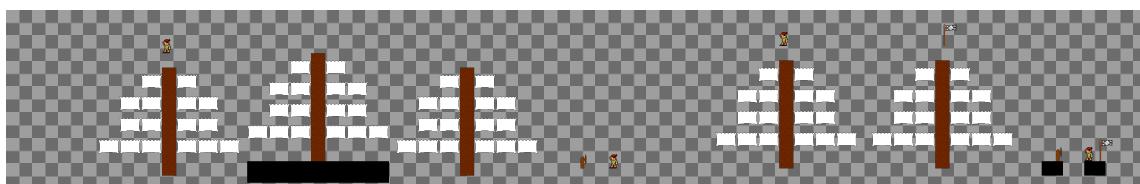


Abbildung 9.49: Hintergrund

**Abbildung 9.50: Hauptebene**

Die obere Abbildung 9.46 zeigt eine Übersicht des Levels „Desert“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.47 der Objektebene, Abbildung 9.48 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.49 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.50 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

**Abbildung 9.51: Übersicht aller Ebenen****Abbildung 9.52: Objektebene****Abbildung 9.53: Vordergrund****Abbildung 9.54: Hintergrund**

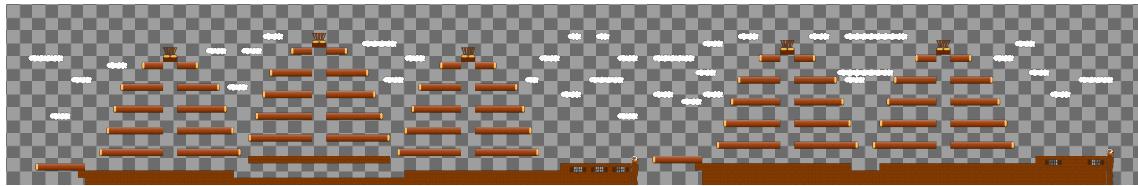


Abbildung 9.55: Hauptebene

Die obere Abbildung 9.51 zeigt eine Übersicht des Levels „Pirates“ mit allen Ebenen. Das Level ist zusammengesetzt aus Abbildung 9.52 der Objektebene, Abbildung 9.53 welche als übergeblendeter Vordergrund erscheint, Abbildung 9.54 welche als zurückgesetzter Hintergrund erscheint und Abbildung 9.55 welche die Spielebene mit Kollisionserkennung darstellt. Der grau gekachelte Hintergrund der Abbildungen ist im Original transparent.

9.4 Umzug auf das finale Betriebssystem

Die Familie der Ubuntu Betriebssysteme ist als Desktop Betriebssystem entworfen. Dies erklärt die Vielzahl an verfügbaren Paketen. Der Nachteil eines Desktopbetriebssystems ist, dass es zusätzlich zu den Game-Pads noch zumindest eine Tastatur, optimal auch eine Maus benötigt. Für den Betrieb ausschließlich mit Game-Pads wird ein anderes Konzept für das Betriebssystem benötigt.

Das in der Präsentation verwendete Betriebssystem Retropie ermöglicht es das volle Frontend (die für den Benutzer sichtbare grafische Oberfläche) des Betriebssystems mittels der Game-Pads zu verwenden. Retropie setzt hierfür auf ein Programm namens Emulationstation. Sowohl Ubuntu als auch Retropie Betriebssysteme basieren auf einem Linux-Kernel, was die Portierung des entwickelten Source Codes und der Konfiguration unnötig macht, da die identischen Pakete verwendet werden können. Es kann somit auf Tastatur und Maus verzichtet werden.

Ein weiterer Vorteil von Retropie ist die einfache Erweiterbarkeit und Anpassungsfähigkeit via Konfigurationsskripten. Nahezu alle Ereignisse am System können via Konfiguration angepasst werden. Dies wird in den späteren Abschnitten genutzt um einige Konfigurationen beim Systemstart und beim Betreten bzw. beim Verlassen der Spiele Ereignisse im System zu steuern.

9.5 Einbinden des Spiels in die Emulationstation Umgebung

Retropie bietet verschiedene Möglichkeiten das separat entwickelte Spiel in das System ein zu binden und im Menü sichtbar zu schalten. Erste Versuche das Spiel direkt aus dem Frontend (Emulationstation) via Skript im Verzeichnis /home/pi/RetroPie/roms/ports mittels der Datei DManWar.sh mit dem Inhalt

```
cd /home/pi/Development/MPMG-Game/RPIGames/DManWar;
./DmanWar
```

zu starten, waren zwar erfolgreich, allerdings ist dies nicht der bevorzugte Weg. Der

Nachteil dieser Konfiguration ist, dass Emulationstation keine eigenen Pre- und Post-Funktionen beim Starten und Verlassen von Programmen aufrufen kann.

Es erwies sich als besser den Standardweg zu benutzen. Hierfür wird in der Datei /home/pi/RetroPie/roms/ports/DmanWar.sh folgender Inhalt verwendet:

```
"/opt/retropie/supplementary/runcommand/runcommand.sh" 0  
_PORT_ "dmanwar" ""
```

Hierfür muss weiter die Datei /opt/retropie/configs/ports/dmanwar/emulators.cfg erstellt werden und mit folgendem Inhalt versehen werden:

```
dmanwar = "cd /home/pi/Development/MPMG-  
Game/RPIGames/DManWar; ./DmanWar"  
default = "dmanwar"
```

Diese Konfiguration führt mittels dem Skript /home/pi/RetroPie/roms/ports/DmanWar.sh die internen Mechanismen von Emulationstation aus und führt den default-Befehl in der entsprechenden Konfigurationsdatei /opt/retropie/configs/ports/dmanwar/emulators.cfg aus.

9.6 Automatisches Setzen des Logos auf den Game-Pads

Beim Starten des Systems und nach dem Verlassen der Programme sind die Displays mit den letzten Informationen des zuletzt ausgeführten Schreibvorgangs gefüllt.

Angepasste Anleitung von

<https://retropie.org.uk/forum/topic/9133/quick-and-easy-guide-for-adding-music-to-emulationstation-on-retropie-noob-friendly>

Das Skript /opt/retropie/configs/all/autostart.sh wird zum Starten des Frontends (Emulationstation) verwendet. Es kann um die notwendigen Bestandteile zum Beschreiben der Displays erweitert werden. Hierfür muss es mittels dem folgenden Befehl geöffnet werden.

```
sudo nano /opt/retropie/configs/all/autostart.sh
```

Um die Displays der angeschlossenen Game-Pads zu beschreiben, muss die Datei folgende Befehle enthalten.

```

for fbs in /dev/fb*
do
  if [ "$fbs" != "/dev/fb0" ]
  then
    (sudo fbi -d $fbs -T 1 -noverbose -a

/home/pi/Development/Treiber/test/Logo$#
  fi
done
cd /home/pi/Development/MPMG-Game/RPIGames/DManWar;

emulationstation #auto

```

Das auf diese Weise veränderte Skript liest nun aus dem Dateisystem die angemeldeten Framebuffer-Geräte aus und verwendet fbi um auf den Displays das ausgewählte Logo (/pi/Development/Treiber/test/Logo.png) zu schreiben. Framebuffer 0 („/dev/fb0“) wird nicht beschrieben, dieses Framebuffer-Gerät stellt die Schnittstelle des Hauptbildschirms dar. Im Anschluss wird wie gewohnt das Emulationstation-Frontend gestartet. Wird der Raspberry Pi 3 nun neugestartet, erscheint das Bild Logo.png auf den Displays.

Um das Logo beim Verlassen eines ausgeführten Programms wieder auf den Displays an zu zeigen, muss die Datei /opt/retropie/configs/all/runcommand-onend.sh mittels der schon oben verwendeten Anweisungen angepasst werden. Die Datei besitzt nun folgenden Inhalt:

```

for fbs in /dev/fb*
do
  if [ "$fbs" != "/dev/fb0" ]
  then
    sudo fbi -d $fbs -T 1 -noverbose -a

/home/pi/Development/Treiber/test/Logo.$#
  fi
done

```

Wenn die Datei nicht existieren sollte, so muss diese erzeugt und mit den korrekten Rechten versehen werden. Sie sollte für alle Benutzer als ausführbar gekennzeichnet werden. Hierfür dient der folgende Befehl:

```

sudo chmod a+x /opt/retropie/configs/all/runcommand-
onend.sh

```

10 Ausblick

Die vorgestellte Hardware bietet die geforderte Funktionalität, diese kann allerdings noch um eine Vielzahl an Funktionen erweitert werden und an einigen Stellen verbessert werden.

So kann zum Beispiel die Vergabe der Adresse mittels der verbleibenden drei Leitungen am Gameport geschehen. Die Adressierung der Game-Pads wird derzeit über den Adressencoder eingestellt. Die drei nicht belegten Leitungen des Busses können verwendet werden um die Adressierung der Game-Pads permanent korrekt zu halten. Für dieses Verfahren müssen jedem Busanschluss am Gameport eine eindeutige Bitmaskierung zugewiesen werden, welche auf der Platine hart auf GND und 3v3 verbunden werden müssen. Die Game-Pads müssen an dieser Stelle komplexer gestaltet werden um die Leitungen vom Busanschluss bis an die Adresspins des MCP23017 zu führen. Die derzeitige Konfiguration ermöglicht diverse Adressierungsarten, welche unabhängig vom verwendetem Steckplatz sind. Die derzeitige Hardware ermöglicht allerdings eine Fehladdressierung, da mehrere Gamapads unter der identischen Adresse angeschlossen werden können.

Bei einer weiteren Überarbeitung des Design sollte die Verdrahtung zwischen Game-Pad und Gameport überarbeitet werden. Eine Verbesserung des Übertragungskabels ermöglicht eine geringere Einstreuung und verringert die Störempfindlichkeit. Auf diese Weise kann die Ausführung des Treibers beschleunigt werden. Anstelle von den derzeit drei aufeinander folgenden Lesezugriffen sollte es möglich sein mit einem einzigen Lesevorgang die Daten vom Game-Pad mit geringer Fehlerrate zu lesen.

Ein weiterer Engpass welcher einer Überarbeitung würdig wäre ist die Übertragung auf dem SPI-Bus. Eine höhere Framerate auf den Displays würde mehr Möglichkeiten für ihren Einsatz bieten.

Für die Verwendung mit Emulationstation und der mit dieser Umgebung gelieferten Standardfunktionen können weitere Änderungen am Betriebssystem sinnvoll sein. So könnten zum Beispiel die Displays verwendet werden, um beim Starten eines Emulators oder eines Spiels das Cover aus dem Menü auf den Game-Pads an zu zeigen. Zudem kommt es bei den Emulatorspielen häufig zu verwirrungen darüber welches Game-Pad für den ersten oder den zweiten Spieler verwendet wird. Dies könnte zusätzlich auf den Displays vermerkt werden. Im Menü könnten die Displays verwendet werden um Zusatzinformationen zu den ausgewählten Spielen oder den Konsolen an zu zeigen.

Diese Projektarbeit bezog sich auf das Kennenlernen der Betriebssystemeigenen Schnittstellen bezüglich Hardware- und Softwarekommunikation. Die Hardware wurde somit erstellt um eben diese Möglichkeiten des Raspberry Pi zu nutzen. Moderne Konsolen besitzen nahezu ausschließlich nur noch kabellose Controller. Für eine kommerzielle Vermarktung sollte der Erfolg dieser Konzepte zu Rate gezogen werden. Würden die Controller eine eigene Stromversorgung, so wie mehr eigene Intelligenz (in Form eines Mikrocontrollers) erhalten wäre es möglich die Game-Pads mittels WLAN oder Bluetooth mit dem Raspberry Pi zu koppeln. Eine auf diese Weise verbesserte Hardware benötigt stark veränderte Treiber, würde allerdings auch das Problem der

Einstreuung auf den Signalleitungen des gemeinsam genutzten Busses verringern. Das derzeitige Game-Padkonzept benötigt zudem ein permanentes Polling der Tasten. Game-Pads, welche wie zuvor beschrieben mit dem Raspberry Pi 3 kommunizieren, können dem Raspberry Pi interruptbasiert ihre Daten zur Verfügung stellen. Auch die Anzahl der teilnehmenden Spieler kann massiv vergrößert werden.

Der entwickelte Treiber zur Displaysteuerung via MCP23017 Portexpander kann in zukünftigen Arbeiten wiederverwendet werden. Dies ermöglicht die sparsame Verwendung der Raspberry Pi eigenen GPIOs und das Betreiben mehrerer Displays am SPI-Bus mittels Adressierung.

Die zusätzlich für das Spiel erledigten Aufgaben im Bereich Grafik- und Leveledesign können selbstverständlich ebenfalls für eine Erweiterung des Spiels sorgen. Ein Spiel mit weiteren Levels, Gegnern und Umgebungsgrafiken würde das Spielerlebnis steigern und könnte zu einem größeren Markterfolg führen. Auch weitere Grafiken für visuelle Effekte können sinnvoll sein. So könnten weitere Hintergrundebenen und Grafiken für Parallax-scrollen verwendet werden. Dies erhöht den Eindruck optischer Tiefe. Auch bewegliche Objekte und deren Pfade könnten in den Levels vorgesehen werden.

Das Ziel dieser Projektarbeit wurde erreicht, doch für eine kommerzielle Vermarktung müssen wie eben beschrieben weiter Herausforderungen gelöst werden. Die Konzepte welche im Vorfeld geplant wurden, sind im benötigten Umfang eingebaut, zukünftige Verbesserungen stehen allerdings aus.

Anhang

Informationen wie die Datenblätter der verwendeten Bauteile können der beigefügten DVD entnommen werden.

Microchip Technology Inc., Portexpander MCP23017: mcp23017.pdf [I26]

Sitronix Technology Co. Ltd., Display-Controller ST7735: ST7735.pdf [I27]

E-Switch Inc, Adressencoder SRT-10c: srt-10c.pdf [I28]

Literaturverzeichnis

A Bücher/Monographien

- [1] Kofler, M., Kühnast, C., Scherbeck, C., Raspberry Pi, Das umfassende Handbuch, 3. Auflage., Rheinwerk Verlag GmbH, Bonn, 2016, S. 443-446
- [2] Kofler, M., Kühnast, C., Scherbeck, C., Raspberry Pi, Das umfassende Handbuch, 3. Auflage., Rheinwerk Verlag GmbH, Bonn, 2016, S. 463-472
- [3] Corbert, J., Rubini, A., Kroah-Hartmann, G., Linux Device Drivers, 3. Auflage., O'Reilly Media Inc., Sebastopol, 2005, S. 1-41
- [4] Corbert, J., Rubini, A., Kroah-Hartmann, G., Linux Device Drivers, 3. Auflage., O'Reilly Media Inc., Sebastopol, 2005, S. 107-123
- [5] Corbert, J., Rubini, A., Kroah-Hartmann, G., Linux Device Drivers, 3. Auflage., O'Reilly Media Inc., Sebastopol, 2005, S. 196-202
- [6] Corbert, J., Rubini, A., Kroah-Hartmann, G., Linux Device Drivers, 3. Auflage., O'Reilly Media Inc., Sebastopol, 2005, S. 213-234

B Internet-Adressen

- [I1] Elektronik-Kompendium.de, Raspberry Pi: Belegung GPIO, <https://www.elektronik-kompendium.de/sites/raspberry-pi/1907101.htm>, Stand: 09.02.2018
- [I2] Arduino, New library for ST7735 displays and the "Minions" TFT on eBay, <https://forum.arduino.cc/index.php?topic=397984.0>, Stand 09.02.2018
- [I3] Bernd Jahnke, Technische Daten und Informationen zur Nintendo Wii U <http://www.zimmer101.de/wiiu-spiele/technik.html>, Stand 09.02.2018
- [I4] Pinterest, Rs.280/-5pcs Mini DC DC 12 24 V To 5V 3A Step Down Power..., <https://www.pinterest.de/pin/360217670185935831/>, Stand 09.02.2018
- [I5] Element14, Raspberry Pi 3 Model B GPIO 40 Pin Block Pinout, <https://www.element14.com/community/docs/DOC-73950/l/raspberry-pi-3-model-b-gpio-40-pin-block-pinout>, Stand 09.02.2018
- [I6] Canonical Ltd., Ubuntu MATE for the Raspberry Pi 2 and Raspberry Pi 3, <https://ubuntu-mate.org/raspberry-pi/>, Stand 09.02.2018
- [I7] Slashdot Media, Win32 Disk Imager, <https://sourceforge.net/projects/win32diskimager/>, Stand 09.02.2018
- [I8] Matt Hawkins, Enable SPI Interface on the Raspberry Pi, <http://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/>, Stand 09.02.2018
- [I9] Anton Hammerschmidt, Raspberry Pi GPIO How-To,

- <http://raspberrypiguide.de/howtos/raspberry-pi-gpio-how-to/>, Stand 09.02.2018
- [I10] Noralf Trønnes, notro/rpi-source, <https://github.com/notro/rpi-source/wiki>, Stand 09.02.2018
- [I11] Bernhard Grotz, Präprozessor, Compiler und Linker, <https://www.grundwissen.de/informatik/c/präprozessor-compiler-linker.html>, Stand 09.02.2018
- [I12] Noralf Trønnes, notro/fbtft, <https://github.com/notro/fbtft>, Stand 09.02.2018
- [I13] SnowBro, Protecting GPIOs from ESD, <https://www.raspberrypi.org/forums/viewtopic.php?t=203408>, Stand 09.02.2018
- [I14] The Imaging Source Europe GmbH, RGB565, https://s2.www.theimagingsource.com/application-0.0.5864/documentation/ic_imaging_control_class/en_US/images/rgb565.gif, Stand 09.02.2018
- [I15] Best-Microcontroller-Projects, The MCP23017 I/O Expander, <http://www.best-microcontroller-projects.com/mcp23017.html>, Stand 09.02.2018
- [I16] Heise Gruppe GmbH & Co. KG, 7"-Touch-LCD für den Raspberry Pi, <https://www.heise.de/ct/ausgabe/2015-21-aktuell-Raspberry-Pi-2815294.html>, Stand 10.03.2018
- [I17] Raspberry Pi Foundation, raspi-config, <https://www.raspberrypi.org/documentation/configuration/raspi-config.md>, Stand 10.03.2018
- [I18] Raspberry Pi Foundation, SSH (Secure Shell), <https://www.raspberrypi.org/documentation/remote-access/ssh/README.md>, Stand 10.03.2018
- [I19] TheGamesDB.net, Mega Man, <http://thegamesdb.net/game/360/>, Stand 10.03.2018
- [I20] TheGamesDB.net, Super Mario World, <http://thegamesdb.net/game/136/>, Stand 10.03.2018
- [I21] TheGamesDB.net, Sonic the Hedgehog, <http://thegamesdb.net/game/5544>, Stand 10.03.2018
- [I22] Tomas Sardyha, Markup, <https://github.com/darsain/motio/blob/master/docs/Markup.md>, Stand 10.03.2018
- [I23] pyrotek7x7, Tileset, <https://www.spriters-resource.com/nes/mm2/sheet/2324/>, Stand 10.03.2018
- [I24] TheGamesDB.net, Mario Bros, <http://thegamesdb.net/game/189/>, Stand 10.03.2018
- [I25] Thorbjørn Lindeijer, Tiled, <http://www.mapeditor.org/>, Stand 10.03.2018

- [I26] Microchip Technology Inc., MCP23017,
<http://ww1.microchip.com/downloads/en/DeviceDoc/20001952C.pdf>
- [I27] Sitronix Technology Co. Ltd., ST7735 Datasheet (PDF),
<http://pdf1.alldatasheet.com/datasheet-pdf/view/326213/SITRONIX/ST7735.html>,
Stand 10.03.2018
- [I28] E-Switch Inc., ROTARY CODED SWITCH,
<http://www.datasheets360.com/pdf/-8366236898164354791>, Stand 10.03.2018