Konzeption und prototypische Implementierung eines Multiuser Spiels unter Nutzung von Sensordaten auf mobilen Geräten

Diplomarbeit

Angewandte Informatik

Fachbereich 4: Wirtschaftswissenschaften II

vorgelegt von:	Carsten Guhl
Matrikelnummer:	s0514436

Erstgutachter:	Prof. Dr. Jürgen Sieck
Zweitgutachter:	Prof. Dr. Thomas Jung

Danksagung

Hiermit bedanke ich mich recht herzlich bei Prof. Dr. Jung und insbesondere bei Prof. Dr. Sieck, der mich indirekt auf das Thema gebracht hat und viele nützliche Tipps während der Arbeit gegeben hat.

Dank geht auch an Eileen Kühn für die Hilfe bei der Implementierung.

Zu guter Letzt danke ich alle Personen in meinem Umfeld, die mich während dieser Arbeit unterstützt haben.

Inhaltsverzeichnis

1. Einleitung

Schon zu Zeiten des Gameboys hatte man die Möglichkeit bei einigen Spielen mit oder gegeneinander zu spielen. Doch damals fand die Kommunikation der Endgeräte über ein Kabel statt. Ferner war die grafische Leistung aus heutiger Sicht ungenügend. Der Spielspaß und die Motivation dieser Spiele bestand immer darin sich entweder auf Highscorelisten zu verewigen oder im direkten Vergleich mit Freunden sein Können zu beweisen. Wenn man sich die Entwicklung der Mobiltelefone in den letzten 10 Jahren anschaut, dann sieht man einen großen Fortschritt in der Entwicklung der Technik. Aus einfachen Handys, die lediglich über die Funktion verfügten Telefonate zu tätigen und Kurzmitteilungen zu versenden, haben sich in den letzten Jahren Alleskönner, so genannte Smartphones entwickelt, welche eine Kombination aus Telefon, PDA, sogar GPS Empfänger und Digitalkameras darstellen. Selbst 3D-Anwendungen sind möglich.

1.1. Ziel der Arbeit

Im Rahmen dieser Arbeit wird die Entwicklung eines Spiels mit dem Namen - *Balls'n'Cubes* - dargestellt. Dies bietet neben einer grafiklastigen 3D-Spielwelt die Möglichkeit sich einen Spielpartner zu suchen. Folglich wird die Möglichkeit offeriert sich in eine Highscoreliste einzutragen zuzüglich der Option seine Positionsdaten hinzuzufügen. Ziel wird es sein, ein System zu entwerfen und prototypisch umzusetzen, so dass eine geeignete Kommunikation im Mehrspielermodus mit dem Gegenspieler gegeben ist. Darüber hinaus wird die Steuerung der Spielfigur mittels eines Schwerkraft bzw. Beschleunigungssensors in mobilen Geräten gegeben sein. Der Kreis der mobilen Geräte wird in dieser Arbeit auf die Mobiltelefone beschränkt sein. Weiterhin wird auf die Performance der Anwendung zu achten sein, sei es durch Netzwerklatenz oder

Hardwarevoraussetzungen, die nicht vergleichbar mit Desktop PCs sind. Das Hauptaugenmerk dieser Abhandlung liegt auf der Realisierung einer performanten Kommunikation sowie der Integration der Sensordaten, um ein Spiel mit hohem Spielspaßfaktor bzw. hoher Useability in der Praxis zu erreichen.

1.2. Aufbau der Arbeit

Anfangs werden im Grundlagenteil die zum Einsatz kommenden Sensoren, sowie Geräteorientierung und Beschleunigung beschrieben. Darüber hinaus wird Grundwissen zu verteilten Systemen, Netzwerken und Protokollen vermittelt. In der Anforderungsanalyse werden dann die mobilen Geräte genauer betrachtet. Welche Leistung erbringen sie? Mit welchen Betriebssystemen laufen sie und welche Hardwarevoraussetzungen müssen sie mitbringen? Weiterhin werden bestehende bzw. verwandte Anwendungen herangezogen um folglich Use-Cases der Anwendung aufzuzeigen. Weitergehend wird dann im Systementwurf auf die Voraussetzungen des Systems eingegangen und es wird untersucht wie sich am besten eine Kommunikation zum Spielpartner in Anbetracht der Anforderungen umgesetzt werden kann. Anhand der Entscheidung folgt dementsprechend eine Beschreibung des Systemaufbaus und auf welchen mobilen Endgerät die Software entwickelt wird. Die Implementierung befasst sich dann weitestgehend mit den Problemen und Lösungen, die bei der Umsetzung des Entwurfs auftraten. Hier wird auf die Realisierung der Kommunikation, der Anwendungslogik, der Datenhaltung sowie auf die grafische Benutzeroberfläche Bezug genommen. Bei der Demonstration werden die Parameter für das Testszenario abgesteckt. Um auf die Benutzbarkeit und Performance Rückschlüsse ziehen zu können, wird eine Bewertung des Systems durchgeführt. Zu guter Letzt gibt es eine Schlussbetrachtung, welche sich mit der Anforderungsanalyse befasst und einen Ausblick zur Weiterentwicklung des Prototypen gibt.

2. Grundlagen

In diesem Kapitel werden die Grundlagen, die zum Verständnis der Arbeit von Bedeutung sind näher erklärt.

Zunächst wird auf die mobilen Geräte eingegangen, um die Grundlagen dieser Arbeit besser abstecken zu können. Des weiteren wird die Funktionsweise von Sensorsystemen erklärt, da sie ein wesentlicher Bestandteil für die Interaktion mit dem Spiel darstellen. Weiterhin werden Netzwerke und die wesentlichen Protokolle untersucht, um Einblicke in die Kommunikation zwischen Netzwerkteilnehmern aufzuzeigen. Anschließend wird auf eine Technik zur Anzeige grafischer Inhalte eingegangen sowie auf die Programmiersprache Objective-C, die zum Teil Abseits von Programmiersprachen wie Java oder C++ steht. Schließlich erfolgt eine Zusammenfassung aus der hervorgehen soll welches das zu nutzende Mobiltelefon für die Arbeit sein wird.

2.1. Mobile Endgeräte

Wie bereits in der Einleitung erwähnt wurde, beschränkt sich der Umfang der mobilen Geräte auf Mobiltelefone, genauer gesagt auf Smartphones. Sie sind laut Definition eine Kombination aus Mobiltelefon und einem PDA und haben ähnlich dem PC-Segment, in Bezug auf Moore's Gesetz[1], eine starke Entwicklung erlebt. Zur Zeit befindet sich die Prozessorleistung in etwa um die 500 MHz. Weiterhin sind eine beachtliche Anzahl an Sensoren in den Geräten, wie Kompass, GPS, Schwerkraftsensor, Näherungssensor, Beleuchtungssensor und berührungsempfindliche Displays eingebaut. Seitdem Apple das iPhone 2G veröffentlicht hat und damit neue Maßstäbe setzte, ist der Markt für Smartphones extrem gewachsen.

[1] Moore's Gesetz sagt aus, dass sich die Anzahl integrierter Schaltkreise bei minimalen Kosten etwa alle 2 Jahre verdoppelt

Auffallend ist, dass die Software für den Erfolg eines Gerätes wichtiger ist als die Hardware. Somit ist ein weiteres Merkmal für Smartphones gegeben, da sie mit unterschiedlichen Betriebssystemen laufen. Zu den wichtigsten Vertretern der Smartphones gehören das iPhone 3G mit dem Betriebssystem iPhone OS[2], das G1, auf dem Googles Android läuft und der Blackberry Storm mit dem Blackberry OS sowie der Palm Pre mit webOS[3] oder das N97 von Nokia auf Basis von Symbian OS. Android und das Symbian OS werden als Betriebssysteme für mehrere Modelle verschiedener Hersteller verwendet, während das iPhone, Blackberry und Palm Pre das Betriebssystem aus dem eigenen Hause benutzen.

Generell kann man sagen, dass es für alle Geräte möglich ist Software zu entwickeln, da für alle Betriebssysteme eine SDK[4] angeboten wird. Die SDK zur Entwicklung für webOS Software befindet sich zur Zeit leider noch im Beta Stadium. Weiterhin gibt es bei der Entwicklung von Software für die Betriebssysteme Unterschiede hinsichtlich der Programmiersprachen. Während beim iPhone ausschließlich mit Objective-C und bei Android und Blackberry OS nur mit Java[5] programmiert wird, kann bei Symbian die Software mit C++, Java oder Python realisiert werden.

Weitere Unterschiede lassen sich in der Bedienung dieser Geräte feststellen. Einige Geräte verfügen über keine physische Tastatur. Hier ist die Eingabe nur mit einer virtuellen Tastatur auf dem Display möglich. Auch bei der Auflösung der Displays sind Unterschiede zu erkennen. Während das iPhone, Palm Pre und das G1 eine Auflösung von 480x320 Pixel bieten, kann das Blackberry Storm 480x360 Pixel aufweisen. Die höchste Auflösung ließ sich beim N97 mit 640x360 Pixel feststellen.

[2] angepaßte Version von MacOS X
[3] Nachfolger des Palm OS
[4] Software Development Kit
[5] Android: geschwindigkeitskritische Bereiche greifen auf C/C++ Bibliotheken zu

2.2. Sensorsysteme

In diesem Kapitel wird genauer auf die Sensoren der Mobiltelefone eingegangen, die für das Spiel benötigt werden. Sensoren sind in der Regel (elektrische) Bauteile zur qualitativen oder quantitativen Messung bestimmter Größen und Eigenschaften.[13]

2.2.1. Lokalisierung

Viele Mobiltelefone, wie z.B. das iPhone 3G verfügen über mehrere Möglichkeiten der Positionsortung [8]. Dementsprechend werden die am häufigsten genutzten genauer erläutert.

Mobilfunkortung: Die Mobilfunkortung geschieht mithilfe der Sendemasten des Telefonnetzes. Da ein Telefonnetz in mehrere Zellen aufgeteilt ist, kann man schon eine ungefähre Positionsbestimmung, anhand der Zell-ID sicherstellen. Die Größe einer Funkzelle ist allerdings sehr unterschiedlich und ist in ländlichen Gebieten mit geringer Bevölkerungsdichte größer als im städtischen Bereich. Somit kann allein die Bestimmung mittels der aktuellen Zelle sehr große Differenzen zur eigentlichen Position aufweisen. Eine Verbesserung ist durch Messung der Zeit, welche die Signale zu den umliegenden Masten benötigen, zu erreichen. Mittels Triangulation kann dadurch eine bessere aber immer noch ungenaue Positionsbestimmung durchgeführt werden.

Assited-GPS: Vereinfacht dargestellt funktioniert GPS, indem der Empfänger von mindestens 3 Satelliten Daten empfängt und durch Trilateration seine Position bestimmt. Der Empfänger muss dazu die Umlaufbahndaten der Satelliten wissen, die eine Gültigkeit von 4 Stunden haben. Bei mobilen Geräten wird der GPS-Empfänger allerdings nur dann eingeschaltet, wenn er gebraucht wird, da er die Batterie in hohem Maße belastet. Somit kann davon ausgegangen werden, dass die Bahndaten nicht mehr gültig sind. Deshalb müssen sie erst wieder neu angefordert werden und das kann bei guten Bedingungen schon 45 Sekunden dauern. Diese Zeit, die gebraucht wird um die erste Positionsbestimmung durchzuführen, nennt man „Time To First Fix“, kurz: TTFF. Um diesen

Wert zu verbessern, werden die Bahndaten dem Telefon entweder via Internet oder über das Telefonnetz übermittelt. Auf diese Art und Weise kann der TTFF-Wert bei idealen Bedingungen auf eine Sekunde reduziert werden.[12]

WPS - Wlan Positioning System: Dadurch dass es im innerstädtischen Bereich eine Vielzahl an Hotspots gibt und diese auch alle kartographiert worden sind, ist auch hier eine Positionsbestimmung möglich. Diese taugt aber nur etwas wenn die Dichte der Hotspots entsprechend hoch ist. Diese Technik funktioniert auf ländlichen Gebieten somit nur sehr schlecht. Die Lokalisierung findet statt, indem aus der Signalstärke der umliegenden Hotspots die Position errechnet wird. Der große Vorteil dieser Variante ist, dass sie auch innerhalb von Gebäuden funktioniert und eine Genauigkeit von 10-20m erreicht werden kann.

XPS: Die Firma Skyhook Wireless entwickelte einen Zusammenschluss der Techniken GPS, WPS und Mobilfunkortung. Die entwickelte Technik wurde als XPS bezeichnet. In Abbildung 2.1 kann man sehr gut erkennen, wie sich die

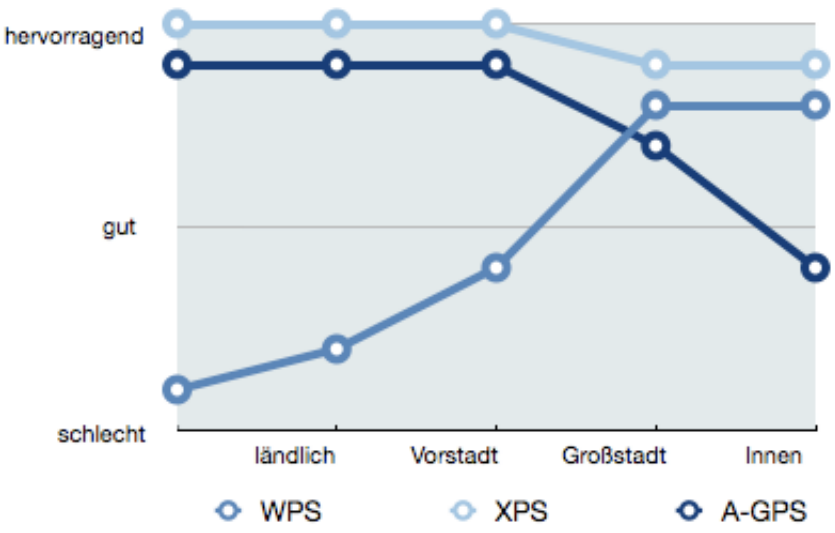

Abbildung 2.1.: Vergleich von Genauigkeit, Verfügbarkeit, TTFF; Quelle: [27]

Vorteile der einzelnen Techniken zu einer ausgereiften Technik der Standortbestimmung entwickelt hat. So kann mit Mobiltelefonen, welche diese Technik verwenden, eine gute Ortung innerhalb kürzester Zeit vorgenommen werden.

2.2.2. Beschleunigungssensor

Beschleunigungssensoren, die in Mobiltelefonen verbaut werden, gehören zu der Klasse der Chip-integrierten Sensoren (MEMS[6]), auch Inertialsensoren genannt. Diese Sensoren können bei einer Größe von nur wenigen Millimetern Beschleunigungen über drei Achsen registrieren und ermöglichen die Messung mit einer Abtastrate von 100 oder 400Hz. [22]

Das Grundprinzip solcher Sensoren beruht auf der Messung der auf eine Siliziummasse wirkende Trägheitskraft. Das Ergebnis wird durch die g-Kraft ausgedrückt. Sofern ein Gerät im Ruhezustand ist entspricht das einer Beschleunigung von 1g[7]. Diese Beschleunigungssensoren sind in der Lage auch Beschleunigungen von mehreren g's auszuhalten.

$$1g = 9,81\frac{m}{s^2}$$

Mit Hilfe eines elektrischen Stroms messen Siliziumfedern die Position der Masse. Bei einer Bewegung kommt es dann in den Federn zu einer Veränderung des Stromflusses. Diese Schwankungen werden schließlich ausgewertet (Piezoelektrizität[8]).

2.2.3. Berührungsempfindliches Display

Um die Berührungen auf einem Display zu erkennen, existieren unterschiedlichste Varianten. Exemplarisch wird eine Variante der Erkennung am Display vom iPhone erklärt, da dieses in der Lage ist auch Multitouch zu registrieren. Dadurch ist dieses Display in der Lage mehrere Berührungen und Bewegungen zur gleichen Zeit zu erkennen und auszuwerten. Aus diesem Grund entstehen neue Eingabemöglichkeiten, welche beispielsweise das Hinein- sowie Herauszoomen in Bilder mit zwei gespreizten Fingern ermöglicht.

Es gibt viele Methoden um Eingaben auf einem Touchdisplay zu erkennen. Die meisten Hersteller benutzen Sensoren und Schaltkreise um Änderungen im Zustand eines Displays zu registrieren. Sie können zwar gut Berührungen

[6] Micro Electro Mechanical System
[7] normale Erdbeschleunigung
[8] beschreibt Änderung der elektrischen Polarisation

in einem Punkt wahrnehmen, aber sobald mehrere Berührungen gleichzeitig erfolgen, ist das Ergebnis fehlerhaft. Einige Displays ignorieren nach der ersten Berührung einfach alle nachfolgenden, oder die Software kann mehrere Berührungen nicht auswerten. Gründe dafür sind, dass unter anderem einige Systeme nur entlang einer Achse oder einer spezifischen Richtung Berührungen erkennen.

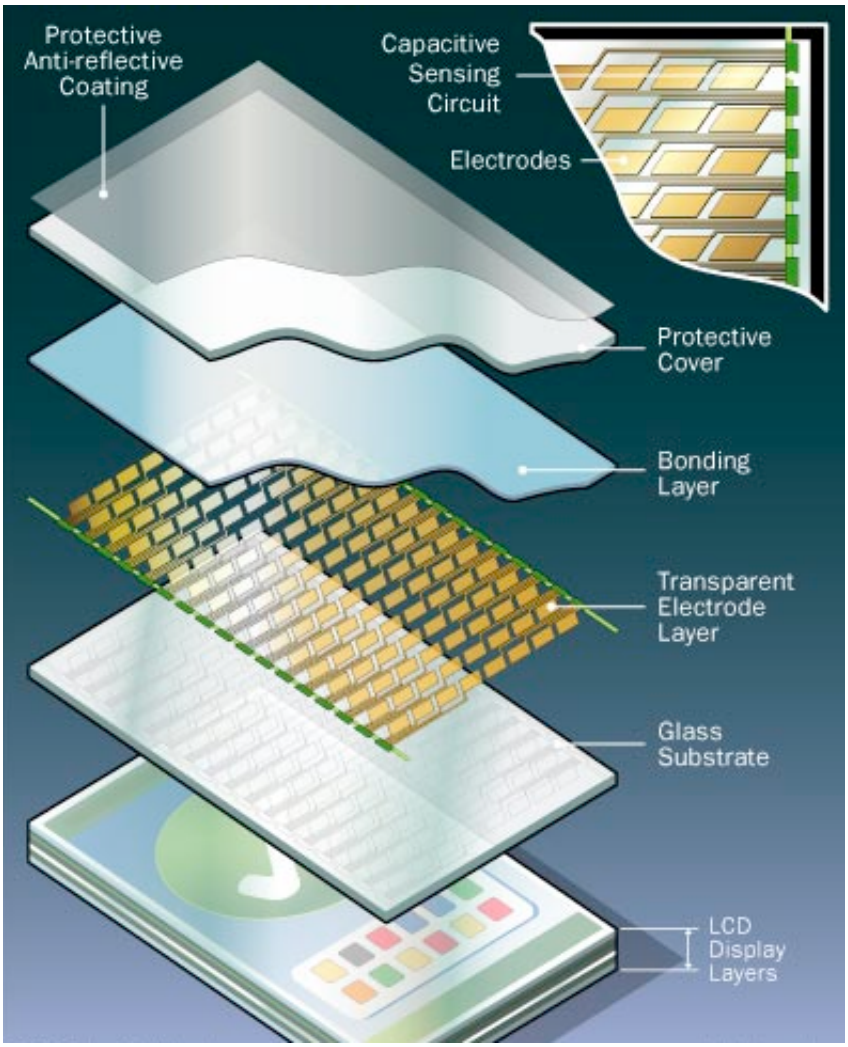

Abbildung 2.2.: Schichten des iPhone Displays; Quelle: [26]

Die Funktionsweise des iPhone besteht darin, Änderungen im elektrischen Strom zu registrieren. Wenn ein Finger auf dem Display platziert wird, ändert es die elektrische Ladung an dieser Stelle in einer Schicht des Displays. Diese Schicht, zwischen LCD-Display und Schutzschicht (Vgl. Abbildung: 2.2),

enthält kapazitive Sensoren[9]. Die Kondensatoren dieser Schicht sind wie ein 2D-Koordinatensystem angeordnet und die Schaltkreise können Berührungen an jedem Punkt des Rasters bemerken. Dabei generiert jeder Punkt sein eigenes Signal, welches dann zum Prozessor weitergeleitet wird. Daraus kann der Prozessor zu jeder Berührung auch eine Bewegung erkennen, durch die Auswertung der Rawdaten (Vgl. Abbildung 2.3). Der Nachteil dieser Technik ist, dass sie bei nichtleitenden Stoffen, wie einem Plastikstift nicht funktioniert.[26]

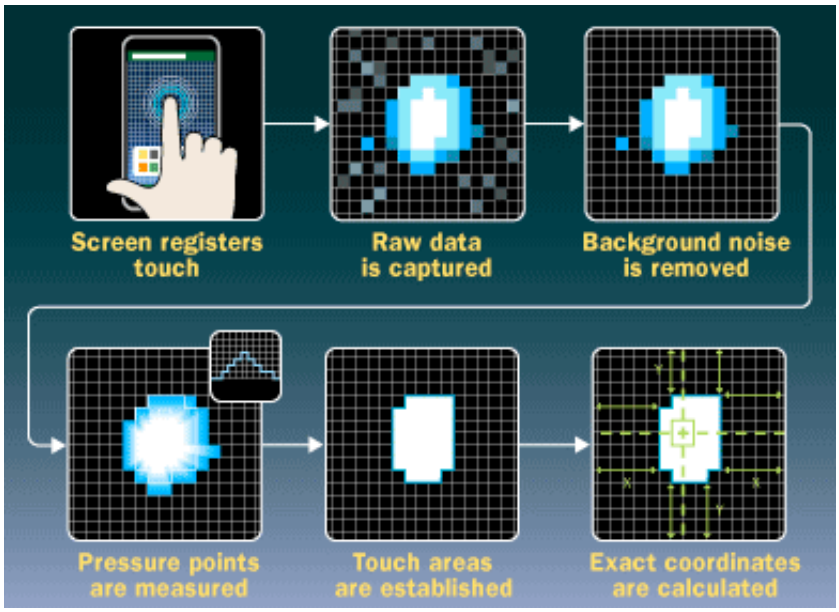

Abbildung 2.3.: Verarbeitung einer Berührung im Prozessor; Quelle: [26]

[9]kapazitive Sensoren werden auch zur Erkennung von Nicht-Metallen verwendet

2.3. Netzwerke und Protokolle

Im Folgenden werden einige Grundvoraussetzungen im Umgang mit Netzwerken erläutert. Diese werden im Hinblick auf die Ausdehnung, Topologie sowie die vorhandenen Architekturen erklärt. Der Zugang zu Netzwerken durch mobile Geräte wird ebenfalls näher beleuchtet. Anhand von Referenzmodellen wird der allgemeine Kommunikationsablauf mit verbindungslosen und verbindungsorientierten Protokollen beschrieben.

> „Ein Computernetzwerk kann man allgemein als Kommunikationsnetzwerk bezeichnen." - A. Schemberg [ScLi07, S. 27]

Hinsichtlich der Kommunikation zwischen Computern und Menschen gibt es keine großen Unterschiede in den Voraussetzungen. Bei beiden gibt es einen Sender und einen Empfänger. Diese unterhalten sich nach gewissen Regeln über ein bestimmtes Medium (Luft, Kabel etc.). Die Nachricht wird dann vom Empfänger verstanden, bzw. decodiert.

Somit reichen zwei Teilnehmer aus, die entweder lokal oder über weite Strecken miteinander kommunizieren, um ein Netzwerk zu bilden.

2.3.1. Netzwerktypen

Den Typ eines Netzwerkes kann man unter anderem anhand der räumlichen Ausdehnung bestimmen (Vgl. Tabelle 2.1). Sobald diese Netzwerke miteinander verbunden werden, bezeichnet man dies als *Internetwork*. [Tane04, S. 41]

Personal Area Network (PAN)

PAN's sind die Netze mit der räumlich kleinsten Ausdehnung. Sie werden in der Regel durch unterschiedliche Übertragungstechniken, wie USB oder Bluetooth, von Kleingeräten aufgebaut. Beispielsweise ist ein Computer, der von seiner Tastatur mittels Bluetooth angesteuert wird, ein PAN.

Entfernung der Prozessoren	Prozessoren im gleichen	Typ
1m	Quadratmeter	PAN
10m	Raum	LAN
100m	Gebäude	LAN
1km	Gelände	LAN
10km	Stadt	MAN
100km	Land	WAN
1.000km	Kontinent	WAN
10.000km	Planeten	Internet

Tabelle 2.1.: Klassifizierung von Netzen nach ihrer Ausdehnung; Quelle: [Tane04, S. 31]

Local Area Network (LAN)

LAN's werden für den Hochleistungsinformationstransfer zwischen gleichberechtigten Teilnehmern, meist über ein Kabel bis zu einer Geschwindigkeit von 10 Gbit/s, benutzt. Die Ausdehnung des Netzwerkes ist innerhalb von Gebäuden oder einem Gelände mit einer maximalen Ausdehnung von bis zu 10km begrenzt. Die kabellose Variante wird als WLAN bezeichnet, die eine Geschwindigkeit von bis zu 54Mbit/s erreicht (Vgl. 2.3.5.1) .

Metropolitan Area Network (MAN)

MAN's breiten sich über mehrere Kilometer aus und sind die sogenannten Stadtnetze. Das deutsche Kabelfernsehen ist ein typisches MAN.

Wide Area Network (WAN)

WAN's können sich über ein sehr großes Gebiet (Länder, Kontinente) erstrecken und werden verwendet um LAN's oder einzelne Rechner zu vernetzen. Dies ist möglich durch Telefongesellschaften oder Internet-Service-Provider. Dadurch ist eine unbestimmte Anzahl von Netzwerkteilnehmern möglich. Um Pakete auch wirklich zum Zielort zu bringen, wird ein Routing-Algorithmus verwendet. Die Systeme, welche die Pakete an die richtige Adresse bringen sind Router, Switches etc..

Global Area Network (GAN)

Bei einem GAN spricht man von der ungbegrenzten geographischen Ausdehnung. Der Begriff wird jedoch selten verwendet. Vielmehr wird vom Internet gesprochen. Obwohl das Internet ein GAN ist, muss nicht jedes GAN Internet genannt werden, da es theoretisch mehrere abgeschottete und unabhängige GAN's geben kann.

2.3.2. Netzwerkarchitekturen

In der Literatur lassen sich zwei Modelle finden die beschreiben, wie Rechner miteinander umgehen und welche Rolle sie einnehmen.

Client-Server-Modell

Ein Client-Server-Modell ist ein Systemdesign, wo die Verarbeitung einer Anwendung in zwei Teile gesplittet wird. Das sind zum Einen der Client und zum Anderen der Server. Der Client gibt den Auftrag für die Bearbeitung von Daten an den Server ab, der dann wiederum das Ergebnis der Arbeit an den Client zurück schickt. Der Server ist somit ein zentrales Mittel für die Bearbeitung von Anfragen von einem oder mehreren Clients (Vgl. Abbildung 2.4). Die Verwendung solch eines Modells findet im LAN statt und spielt auch eine zentrale Rolle im WWW[10]. Viele Kommunikationsprotokolle arbeiten nach diesem Frage-Antwort-Prinzip. Weiterhin hat der Ansatz auch große Verwendung in verteilten Anwendungen. Beispiele für dieses Prinzip, sind z.B. ein Fileserver in einer Firma, der Unmengen von Dateien beherbergt und diese auf Anfrage herausgibt, sowie die Nutzung des WWW mittels eines Browsers. Der Browser (Client) stellt eine Anfrage an einen Webserver (Aufruf einer Seite), der daraufhin mit dem Inhalt der Seite antwortet.

[10] World Wide Web

Abbildung 2.4.: Client-Server-Modell; Quelle: [23]

Peer-To-Peer (P2P)

Den Begriff „Peer-To-Peer“[11] kennt man sicherlich aus der Zeit als Napster[12] mit 50 Mio. Nutzern die Musikindustrie in Aufruhr versetzte. Napster ist somit das berühmteste Beispiel eines P2P-Netzwerkes, obwohl es laut Definition kein echtes P2P ist, da es ein zentrales Dateiregister gab. Denn Teilnehmer in einem P2P sind gleichrangige Instanzen, die selbst organisierend sind. Das heißt, dass es keine zentrale Steuerung oder Hierarchie gibt (Vgl. Abbildung 2.5). Das gegenseitige Finden von Ressourcen bzw. Teilnehmern wird durch ein Overlaynetzwerk[13] erreicht. Die Teilnehmer in solch einem Netzwerk können dann direkt miteinander kommunizieren. Sie können Dienste des P2P nutzen oder eigene Dienste anbieten. Dadurch verfügen sie über einen Zugang zu mehr Ressourcen (Inhalt, Speicher, CPU, Bandbreite), da diese gepoolt werden. [PeDa07]

[11] Peer zu dt.: Gleichgestelllter
[12] Musiktauschbörse
[13] logisches Netz das auf bestehendes Netzwerk mit Infrastuktur aufsetzt

Abbildung 2.5.: Peer-To-Peer-Modell; Quelle: [25]

2.3.3. Referenzmodelle

Um eine Kommunikation zwischen zwei Computer aufzubauen reicht die Hardware allein nicht aus, belegt durch folgendes Zitat:

> „Grundlegene Netzwerkhardware setzt sich aus Mechanismen zusammen, die Bits von einem Punkt zu einem anderen übertragen können. Kommunikation ausschließlich mit Hardware ist aber vergleichbar mit Programmierung, bei der nur Einsen und Nullen eingegeben werden.“ - [Come01]

Folglich ist eine Protokollsoftware[14] erforderlich, die mit der Hardware zusammen arbeitet und ein leistungsstarkes Kommunikationssystem bereitstellt. Um zu gewährleisten, dass die unterschiedlichen Protokolle auch gut zusammenarbeiten, werden sie durch ein Schichtmodell in einen Gesamtplan gesteckt. Dienste bilden eine Schnittstelle zwischen den Schichten. Genauer gesagt kann eine Schicht den Dienst der darunterliegenden Schicht annehmen und stellt dann ihren Dienst der nächsthöheren zur Verfügung.

2.3.3.1. OSI-Referenzmodell

Das OSI-Referenzmodell ist solch ein Schichtmodell und beschreibt im Wesentlichen die Aufteilung des Kommunikationsproblem in Schichten.

[14]Protokoll: Reihe von Regeln (Syntax, Parameter, Eigenschaften)

Folgende Grundgedanken haben zur Anzahl der Schichten beigesteuert:

- Neue Schicht, wenn ein neuer Abstraktionsgrad benötigt wurde
- Möglichst genau definierte Funktion pro Schicht
- Berücksichtigung international genormter Protokolle bei Funktionswauswahl
- Möglichst geringer Informationsfluss zwischen den Schichten

Somit sind letztendlich 7 Schichten, die aufeinander aufbauen, entstanden (Vgl. Abbildung: 2.6):

Die Bitübertragungsschicht sitzt direkt über dem physikalischen Übertragungsmedium auf und regelt die Übertragung von reinen Bits.

Die Sicherungsschicht dient dazu den Bitdatenstrom in Frames[15] aufzuteilen und außerdem fügt sie Sequenznummern und Prüfsummen hinzu, um die fehlerfreie Übertragung zu gewährleisten.

Die Vermittlungsschicht regelt die Verbindung zwischen heterogenen Netzen. Zu den Aufgaben gehören unter anderem die Auswahl der geeigneten Route für ein Paket.

Die Transportschicht ist die Schicht, die oberhalb des Verbindungsnetzes (Schicht 1-3) steht. Das bedeutet, dass hier eine Endpunkt-zu-Endpunkt Kommunikation zwischen Empfänger und Sender vorhanden ist. Sie segmentiert gegebenenfalls Daten von der Sitzungsschicht und stellt sicher das die Daten auch auf der anderen Seite ankommen.

Die Sitzungsschicht ermöglicht den Aufbau einer Sitzung. Deshalb sind Dienste, welche die Synchronisation, den Dialog, und die Wiederaufnahme nach Unterbrechung einer Kommunikation regeln, in dieser Schicht untergebracht.

Die Darstellungsschicht ist ein Übersetzter zwischen den verschiedenen Datenformate (z.B. ASCII). Sie bringt die systemabhängigen Daten in eine unabhängige Form. Somit ist sie weniger mit den Bits einer Übertragung beschäftigt als vielmehr mit dem Syntax und der Semantik.

[15] dt.: Rahmen, Blöcke

Die Anwendungsschicht ist die oberste Schicht und enthält eine Vielzahl von Protokollen und verschafft der Anwendung Zugriff auf das Netzwerk.

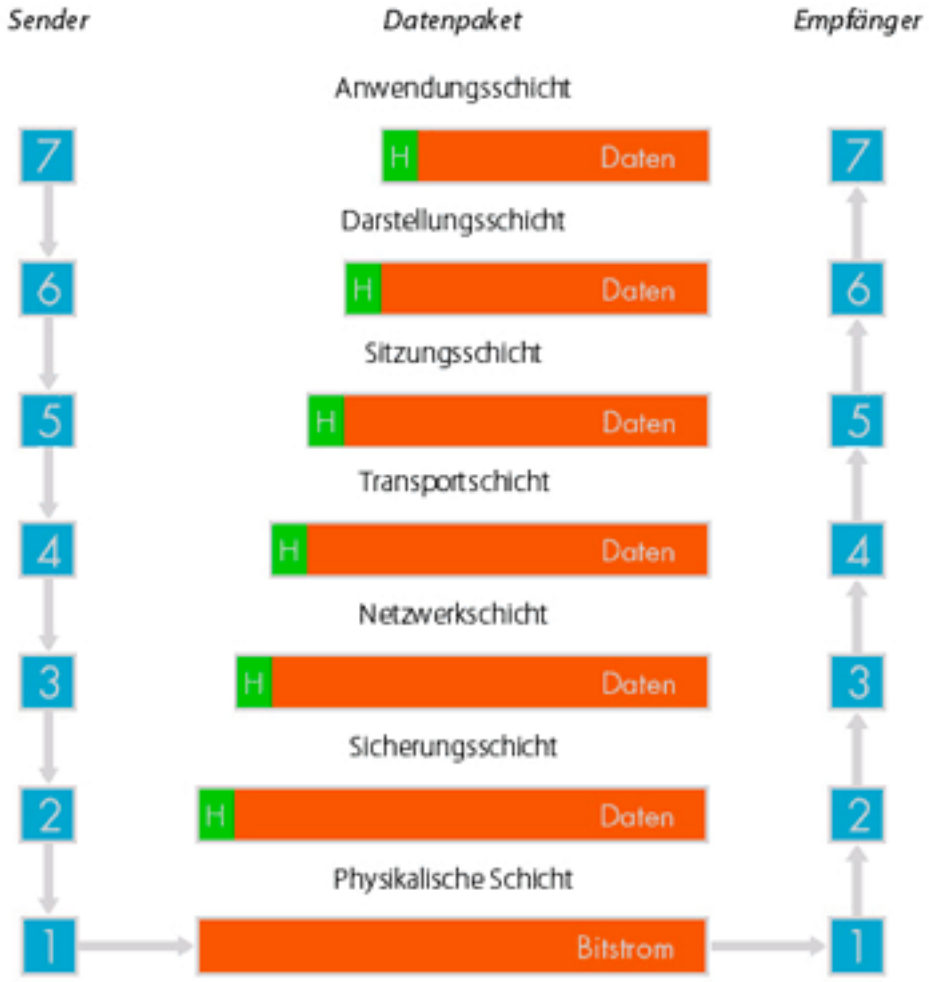

Abbildung 2.6.: OSI-Referenzmodell; Quelle: [20]

In Abbildung 2.6 wird weiterhin der Ablauf einer typischen Kommunikation dargestellt. Sie zeigt wie Daten von der Anwendungsschicht (linke Seite) durch alle Schichten geschickt werden und auf der rechten Seite (Empfänger) von unten nach oben bis zur Anwendungsschicht gelangen. Dabei werden in jeder Schicht die Nutzdaten mit einen zusätzlichen Header versehen oder es wird einer entfernt, abhängig davon ob man entweder Empfänger oder Sender ist.

2.3.3.2. TCP/IP-Referenzmodell

Als ein weiteres Modell ist noch TCP/IP zu erwähnen, da es im Internet eingesetzt wird. Wie aus Abbildung 2.7 zu entnehmen ist, hat dieses Schichtenmodell nur vier Schichten. Die Methodik hinter den Schichten verfolgt den ähnlichen Ansatz wie das OSI-Referenzmodell. Einige Unterschiede sind dennoch vorhanden. So entspricht die Netzzugangsschicht beim TCP/IP-Modell der Bit- und Sicherungsschicht. Darüber hinaus wurden die Sitzungs- und Präsentationsschicht weggelassen, da für diese Schichten meistens kein Bedarf besteht.

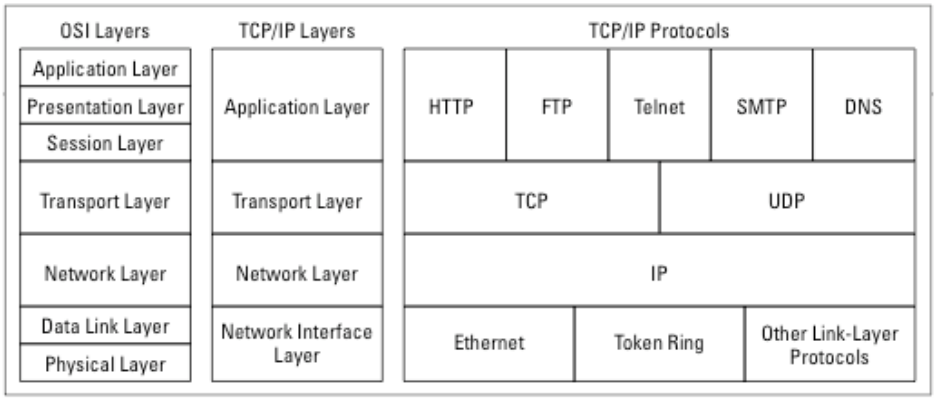

Abbildung 2.7.: TCP/IP im Vergleich zu OSI mit Protokollen; Quelle: [Lowe05, S. 36]

2.3.3.3. Zusammenfassung

Die beiden Modelle haben Vieles gemeinsam, denn sie basieren auf einem Schichtenkonzept unabhängiger Protokolle. OSI unterscheidet im Gegensatz zu TCP/IP stark zwischen den Diensten, Schnittstellen und Protokolle. Als Folge sind die Protokolle bei OSI besser in den Schichten verborgen. Außerdem gibt es Unterschiede in der verbindungslosen und verbindungsorientierten Kommunikation. In der Vermittlungsschicht von OSI werden beide Varianten unterstützt, in der Transportschicht hingegen nur die verbindungsorientierte. Bei TCP/IP ist in der Vermittlungsschicht nur die verbindungslose Kommunikation verfügbar, wohingegen in der Transportschicht sowohl verbindungslose

als auch verbindungsorientierte möglich ist. Diese Auswahl ist wichtig für einfache Frage-Antwort Protokolle.

Das OSI-Modell wurde nie Wirklichkeit, da OSI außerordentlich komplex ist und dadurch eine schlechte Implementierung einherging. Ein weiterer Nachteil war es, dass einige Funktionen wie Adressierung, Flusskontrolle und Fehlerüberwachung mehrmals auftauchten, und somit als ineffektiv angesehen wurde. TCP/IP dagegen hatte eine gute Implementierung und war zusätzlich kostenlos und aus diesem Grund an Universitäten und in Forschungseinrichtungen schon weit verbreitet. Obwohl das Modell praktisch gar nicht richtig existiert, werden die Protokolle häufig angewendet. Das OSI-Modell eignet sich dennoch hervorragend für die Diskussion von Rechnernetzen.

2.3.4. Kommunikationsprotokolle

Im letzten Kapitel wurde die Transportschicht in den Referenzmodellen angesprochen. Sie ist dafür zuständig eine Endpunkt-zu-Endpunkt Verbindung mit dem Empfänger aufzubauen. Da diese Schicht bei der Kommunikation entscheidend ist, werden die Protokolle dieser Schicht im TCP/IP-Modell genauer erläutert.

Um eine Verbindung anhand dieser Protokolle aufzubauen bedarf es einer Softwareschnittstelle. Diese Kommunikationsschnittstelle bilden die Sockets, die vom Betriebssystem bereitgestellt werden. Ein Socket wird beschrieben durch die IP-Adresse, den Port und die Adressfamilie. Durch Verwendung der AF_INET Adressfamilie hat man die Wahl zwischen dem verbindungsorientierten TCP oder dem verbindungslosen UDP.

2.3.4.1. Transmission Control Protocol (TCP)

Das TCP-Protokoll ist ein optimierter, verbindungsorientierter und zuverlässiger Byte-Stromdienst. Es befreit somit Anwendungen sich um fehlende oder unsortierte Daten zu kümmern. TCP benötigt die Phasen des Verbindungsaufbaus und -abbaus durch Drei-Wege-Handshake, um den virtuellen Kanal zwischen Sender und Empfänger herzustellen bzw. zu beenden.

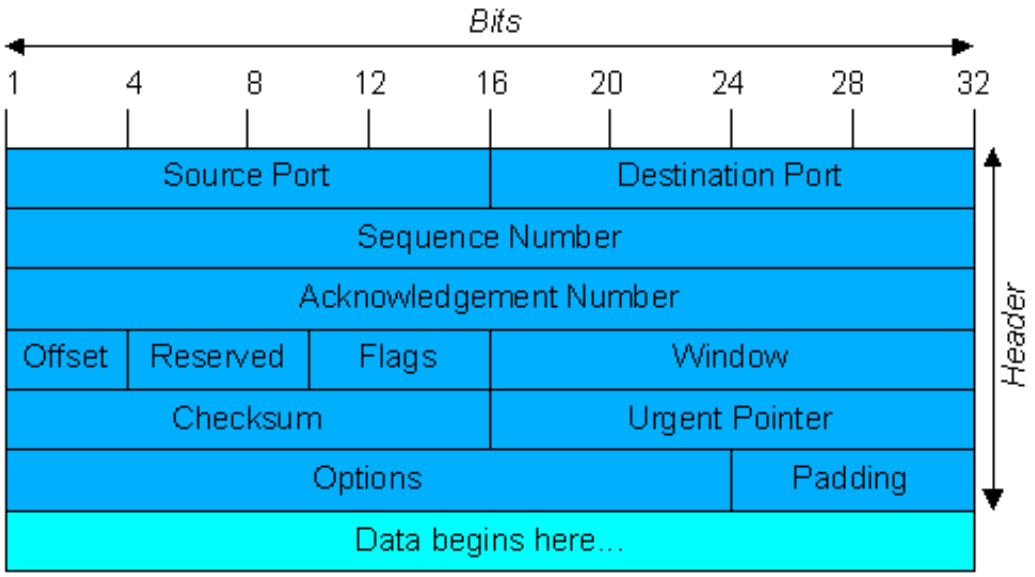

Abbildung 2.8.: Aufbau des TCP-Headers; Quelle: [11]

Weiterhin ist es Vollduplex[16] fähig. Im Header eines TCP-Paketes (Vgl. Abbildung: 2.8) sieht man mehrere Elemente, die zur zuverlässigen Übertragung benötigt werden. Die Flusskontrolle ist solch eine Funktion, die dem Empfänger ermöglicht den Datenstrom vom Sender zu begrenzen. Dies wird mittels einem Sliding-Window Mechanismus erreicht, der auch auf unterschiedliche RTT[17] Rücksicht nehmen muss. Des weiteren gibt es die Überlastkontrolle, die die Geschwindigkeit der Daten drosselt um einer Überlastung des Netzwerkes vorzubeugen.

2.3.4.2. User Datagram Protocol (UDP)

Das UDP-Protokoll ist das einfachste Transportprotokoll der Transportschicht. Der Grund für die Einführung war die bestehende Notwendigkeit eines Protokolls für Sprachübertragungen, da durch die Sicherung der Übertragung von TCP Verzögerungen aufgetaucht wären. So entstand 1977 UDP als nicht zuverlässiges und verbindungsloses Protokoll. Aufgrund der Tatsache, dass nicht erst eine Verbindung aufgebaut werden musste, konnten Sender und Empfän-

[16] gleichzeitiges Senden und Empfangen
[17] RTT: Round-Trip-Time: Zeit vom Sender zum Empfänger und wieder zurück, die für ein Paket benötigt wird

ger schneller mit dem Datenaustausch beginnen. Durch das Weglassen von Sicherungsdiensten, wie z.B. der Flusskontrolle, konnte eine höherer Datendurchsatz und die Netzwerkbelastung gering gehalten werden. Nachteilig ist, dass Pakete nicht ankommen können, sie mehrmals oder nicht in der Reihenfolge ankommen, in der sie losgeschickt wurden. In Abbildung 2.9 kann man erkennen, dass wenig im UDP-Header implementiert ist. Durch die Prüfsumme im UDP-Header wird lediglich sichergestellt, dass wenn ein Paket eintrifft, dieses auch korrekt ist. Weiterhin kann man Zielport und Quellport aus dem Header entnehmen.

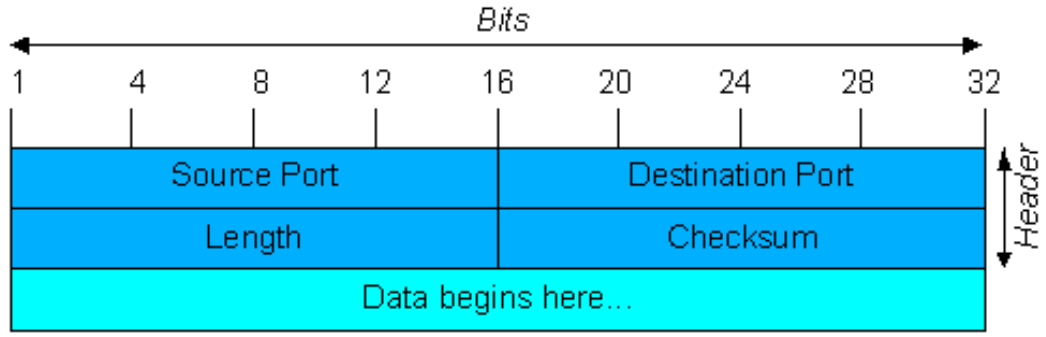

Abbildung 2.9.: Aufbau des UDP-Headers; Quelle: [11]

2.3.5. Drahtlose und mobile Netze

Drahtlose und mobile Netze haben in den letzten Jahren eine große Entwicklung zurückgelegt. Dies hat die Ursache, dass durch das Internet ganz andere Anwendungen, über die Sprache hinaus, an Bedeutung gewonnen haben. Heutzutage sind viele Standards auf dem Markt anzutreffen, die sich auf vielerlei Art von einander unterscheiden. Zum Einen können sie sich in der Bandbreite und Entfernung der Knoten unterscheiden, zum Anderen in der Leistung, die sie verbrauchen. Um hohe Verfügbarkeit, Qualität und Bandbreite zu erreichen, wird viel unternommen, sowohl im Häuslichen als auch im Weitverkehrsbereich. Insgesamt unterscheidet man drei Mobilitätsstufen:

Keine Mobilität Der Empfänger muss an einem festen Standpunkt bleiben um zu empfangen

Eingeschränkte Mobilität Die Mobilität ist nur innerhalb eines bestimmten Bereiches gewährleistet, z.B. bei Bluetooth

Mobil zwischen Basisstationen Mobilität zwischen Basisstationen, z.B. bei Wi-Fi oder Mobiltelefonen

Insgesamt kann man vier große Gruppen der drahtlosen Technologien unterscheiden (Vgl. Tabelle: 2.2).

	Bluetooth 802.15.1	**Wi-Fi 802.11**	**WiMAX 802.16**	**Mobilfunk**
Typische Verbindungslänge	10m	100m	10km	zig km
Typische Bandbreite	2,1 Mbits/s	54 Mbits/s	70 Mbit/s	384+ Kbit/s
Typische Nutzung	Verbindung Peripheriegerät mit Notebook	Verbindung Notebook mit verkabelter Basis	Verbindung Gebäude mit verkabelter Hochantenne	Verbindung Mobiltelefon mit verkabelter Hochantenne
drahtgebundenes Gegenstück	USB	Ethernet	Koaxialkabel	DSL

Tabelle 2.2.: Übersicht führender drahtloser Technologien; Quelle: [PeDa07, S. 134]

Aus der Tabelle ist zu entnehmen, dass alle drahtlosen Varianten dennoch eine verkabelte Basisstation haben. Im Folgenden wird allerdings nur näher auf Wi-Fi und Mobilfunknetzen eingegangen werden, da sie eine Grundlage für die Arbeit bilden.

2.3.5.1. Wi-Fi / WLAN

Wi-Fi ist ein von der Wi-Fi Alliance[18] zertifizierter Standard für den IEEE-Standard 802.11[2]. In vielen Ländern der Welt wird Wi-Fi auch als Synonym für WLAN verwendet, da meistens in einem WLAN ebenfalls der Funkstandard 802.11 verwendet wird. Anhand der Tabelle 2.3 kann man erkennen, dass bereits eine Reihe von Spezifikationen für den Standard herrausgebracht wurden.

[18] Zusammenschluß von mehr als 300 Firmen in 20 Ländern

Arbeistgruppe	Arbeitsgebiet
802.11	Urform des WLAN von 1997, mit 2 Mbits/s im 2,4-GHz-Band
802.11a	54-Mbits/s-WLAN im 5-GHz-Band
802.11b	11-Mbits/s-WLAN im 2,4-GHz-Band
802.11g	54-Mbits/s-WLAN im 2,4-GHz-Band
802.11h	54-Mbits/s-WLAN im 5-GHz-Band mit DFS und TPC
802.11n	300-Mbits/s-WLAN im 2,4-GHz-Band

Tabelle 2.3.: Standards im Bereich WLAN; Quelle: [ScLi07, S. 52]

Einer der gebräuchlichsten Standards ist heute 802.11 b/g, da g abwärtskompatibel zu b ist. Mit dieser Technologie ist auch ein LAN leicht erweiterbar, da die Sicherungsschicht kompatibel zum Ethernet gestaltet wurde, obwohl es große Veränderungen in der Bitübertragungs- und Sicherungsschicht gibt. Dies hat den Grund, dass bei Funknetzen die herkömmlichen Zugriffsverfahren auf das Medium nicht funktionieren. Normalerweise wird gesendet, wenn auf dem Kanal keine Datenübertragung stattfindet. Aber wenn sich, wie in Abbildung 2.3, „Wally“ und „Beaver“ nicht sehen können, kommt es zu Kollisionen, wenn beide an „Ward“ senden wollen. Dieses Problem wird durch ein Algorithmus namens MACA[19] gelöst, indem Steuerframes miteinander ausgetauscht werden.

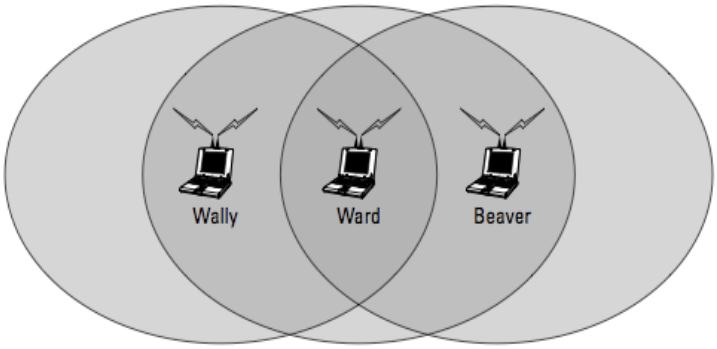

Abbildung 2.10.: Überlappung von Funknetzen; Quelle: [Lowe05, S. 439]

Um ein WLAN aufzubauen wird eine Basisstation benötigt, welche die Koordination der Teilnehmer (Netzknoten) übernimmt. Gibt es keine Basisstation wird dies als Ad-hoc Netzbetrieb bezeichnet, wo die Kommunikation der Teilnehmer direkt stattfindet. Die Basisstationen senden in Intervallen sogenannte Beacon-Frames aus. Diese besitzen den Netzwerknamen, die möglichen Über-

[19]Multiple Access with Collision Avoidance

tragungsraten und Art der Verschlüsselung als Inhalt. Deshalb können sich Netzknoten und Basisstation bei schlechten Bedingungen auf niedrige Übertragungsraten einigen. Es besteht weiterhin ein großer Unterschied zwischen der Sende und Empfangsleistung einiger Geräte. Weiterhin kommen Eigenschaften der Wände, welche die Empfangsleistung erheblich beeinträchtigen können, hinzu. Stahlbetonwände sind solch ein Faktor, der den Empfang äußerst minimieren kann. Ferner stören sich die WLAN's auch gegenseitig. Anhand einer möglichen Kanalwahl kann man sich der Störung durch andere Netze entziehen. Wobei die in Deutschland verfügbaren 13 Kanäle einen Abstand von 5 Mhz haben. So dass nur Kanal 1, 6 und 11 als überlappungsfrei gelten.

Da die kabellosen Daten von Jedem empfangen werden können, ist auch die Sicherheit von bedeutender Rolle. Die WEP-Verschlüsselung[20] kann mittlerweile als nicht mehr sicher angesehen werden, da sie leicht zu knacken ist indem die Schwachstellen der Verschlüsselung, der konstante Schlüssel und die fehlerhafte Integritätsprüfung, ausgenutzt wurden. Die derzeitigen Standards zur Verschlüsselung sind WPA[21] und WPA2.

2.3.5.2. Mobilfunk-Technologien

Neben der Telefonie und dem Versand von SMS sind auch Datendienste, die wiederum auch kostenpflichtig sind, kommerziell verfügbar. Obwohl sich die Frequenzbänder auf der Welt unterscheiden können, sind die Geräte heutiger Generation in der Lage mit vielen dieser Frequenzbänder zu arbeiten. So liegen die Frequenzen in Europa bei 900Mhz und 1800 Mhz, während in Nordamerika 850Mhz und 1900Mhz verwendet werden. Ähnlich wie bei Wi-Fi, stützt sich auch der Mobilfunk auf die Zelltechnologie mit Basisstation. Ein Mobilfunkmast bedient daher immer eine Zelle, oder kann mit mehreren gerichteten Antennen mehrere Zellen bedienen. Ein Telefon meldet sich stets beim Mobilfunkmast mit der höchsten Signalstärke an und steht dann bei diesem unter Kontrolle. Sobald sich das Telefon in eine andere Zelle bewegt, wobei die Grenzen der Zellen nicht scharf sind und daher auch überlappen können, gibt der

[20] Wired Equivalent Privacy
[21] Wi-Fi Protected Access

aktuelle Mobilfunkmast seine Zuständigkeit an den Mobilfunkmast mit der höchsten Signalstärke ab.

Es gibt jedoch keinen einheitlichen Standard für die Datendienste des Mobilfunks, sondern ist eher eine Ansammlung von konkurrierenden Technologien, die nach Generationen eingeteilt wurden. Die erste Generation[22], die aber nicht von Interesse für die Datenkommunikation ist, aufgrund der analogen Übertragung, wird als G1 bezeichnet. Als nächstes ist demnach die zweite Generation zu nennen, wobei der GSM[23]-Standard, weltweit in über 200 Ländern mit mehr als über 1.5 Milliarden Nutzern eingesetzt wird. Auch GSM wurde für die Sprachübertragung konzipiert und ist daher wenig zu gebrauchen. Erst durch die Einführung von GPRS[24] bzw. der Erweiterung von GSM durch EDGE[25] war eine anständige Datenübertragung gewährleistet. Diese Technologie wird auch als 2.5G-Technologie bezeichnet[PeDa07]. Das generelle Prinzip von GPRS ist, dass durch Zeitmultiplexverfahren (TDMA[26]) für eine Verbindung in einem bestimmten Abstand Zeitschlitze zur Verfügung stehen. Da auch die reine Sprachtelefonie Zeitschlitze verbraucht, kann in stark frequentierten Gebieten die Übertragungsrate von GRPS deutlich auf 30-70Kbit/s sinken, weil der Abstand der Zeitschlitze größer wird. Bei guten Bedingungen können Datenraten von bis zu 170Kbit/s erreicht werden..

Die dritte Generation (3G) steht hauptsächlich für den UMTS[27]-Standard, der durch Versteigerung der Lizenzen in Deutschland an die Mobilfunkbetreiber für ca. 50 Mrd. Euro verkauft wurde. Diese Summe bewegte den damaligen Finanzminister Hans Eichel zu folgender Aussage:

> „UMTS stehe für: Unerwartete Mehreinnahmen zur Tilgung von Schulden“ - *Hans Eichel*; Quelle: [18]

Mit der UMTS-Technologie und deren Erweiterung HSDPA[28] soll es möglich sein bis zu 13,98Mbit/s zu erreichen. Da aber die Datenrate von der Kategorie des Empfängers beschränkt ist, werden im Allgemeinen Butto-Übertragungsraten

[22] in Deutschland das A-, B- und C-Netz
[23] Global System for Mobile Communications
[24] General Packet Radio Service
[25] Enhanced Data Rates for GSM Evolution
[26] Time Division Multiple Access
[27] Universal Mobile Telecommunications System
[28] High Speed Downlink Packet Access

von 3,6Mbit/s und 7,2Mbit/s erreicht. Das iPhone 3G befindet sich in der Kategorie mit Übertragungsraten von 3,6Mbit/s. Gegenüber den starren Zeitschlitzen beim GSM-Standard wird bei UMTS ein Verfahren namens CDMA[29] verwendet, wodurch sich die vorhandenen Frequenzen wirtschaftlicher nutzen lassen. Dieses Verfahren erlaubt die gleichzeitige Übertragung verschiedener Datenströme dadurch, dass sie mit unterschiedlichen Codes versehen werden.

2.3.6. Topologien

Die Topologie[30] eines Netzwerkes beschreibt die physikalische Ausdehnung der Netzwerkteilnehmer, entweder durch Kabel oder Funk. Anhand unterschiedlicher Topologien entstehen unterschiedliche Eigenschaften, welche wiederum die einzusetzende Hardware und Zugriffsmethoden bestimmen. Das hat schließlich Einfluss auf die Übertragungsgeschwindigkeit und den Durchsatz der Daten auf dem Medium, wie z.B. ein Kabel.

Weiterhin lassen sich Topologien nach ihren Dimensionen einteilen. Bei lokalen Netzwerken werden in erster Linie eindimensionale Topologien verwendet. Da diese Arbeit auf drahtlose Verbindungen abzielt, können kabelgebundene Topologien, wie Bus oder Ring vernachlässigt werden. Somit gilt das besondere Augenmerk den eindimensionalen Topologien, insbesondere der Stern-Topologie. Die Namen der Topologien beziehen sich weniger auf die physische Anordnung, als vielmehr auf die logische Anordnung.

2.3.6.1. Stern-Topologie

Bei der Stern-Topologie sind alle Netzwerkteilnehmer separat an einem zentralen Gerät, dem Hub oder Switch, angeschlossen (Vgl. Abbildung 2.11). Auf diese Art und Weise besitzt jeder Teilnehmer seine eigene Verbindung zum Hub. Die Stern-Topologie ist heute der typische Vertreter für das Ethernet und WLAN. Bei der drahtlosen Variante halten alle Teilnehmer eine direkte Verbindung über Funksignale zur Basisstation [ScLi07].

[29] Code Division Multiple Access
[30] wörtlich: Geometrie der Lage

Vorteile:

- Bei Ausfall eines Teilnehmers, bleibt das Netz davon unbetroffen
- Leicht erweiterbar
- Einfache Kontrolle

Nachteile:

- Bei Ausfall des Hubs -> Totalausfall
- Bei Kabelnetzwerk viel Verkabelung

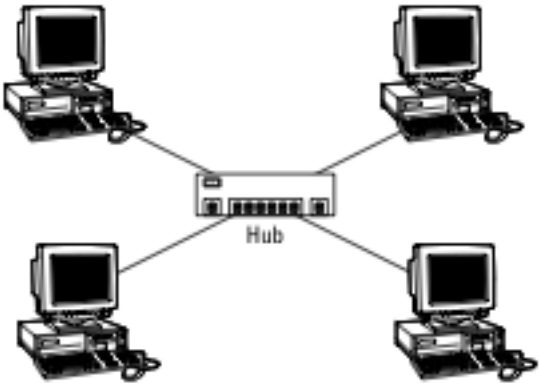

Abbildung 2.11.: Stern-Topologie im verkabelten Netzwerk; Quelle: [Lowe05, S. 16]

2.3.7. Bonjour

Fast alle Geräte, die in einem Netzwerk vorkommen können, verstehen sich über das IP-Protokoll. In lokalen Netzwerken kann es aber dennoch zu Problemen kommen. Damit IP überhaupt arbeiten kann, benötigt jedes Gerät in einem Netzwerk eine eindeutige IP-Adresse, entweder dynamisch über DHCP zugeteilt oder statisch vordefiniert. Sollen Dienste per Name angesprochen werden, muss außerdem noch ein DNS-Server, der die Übersetzung von Namen in IP-Adressen vollführt, konfiguriert sein. Um ein Gerät aus einem Netzwerk zu benutzen, muss es mit dem Namen oder der IP-Adresse angesprochen werden. Somit muss der Benutzer immer den Namen seiner Geräte in einem Netzwerk wissen, um diese anzusprechen.

Diese Problematik kann durch Bonjour[31], wie das folgende Zitat belegt, gelöst werden.

> „People need a simple and reliable way to configure and browse for services over IP networks. They want to discover available services and choose one from a list, instead of having to know each service's name or IP address in advance. It is in everyone's interest for IP to have this capability. This is exactly the capability that Bonjour provides.“ - *Apple Computer, Inc.*

Somit ist Bonjour, eine Zeroconf[32]-Netzwerkarchitektur, die das einfache Bereitstellen und Finden von Netzwerkdiensten in einem IP-Netzwerk ermöglicht. Um die Zeroconf-Spezifikation zu erfüllen, müssen folgende durch die ZERO-CONF Working Group[33] definierten Eigenschaften gegeben sein.

- **Automatische Zuweisung von IP-Adressen ohne DHCP:** Das Gerät wählt selbständig eine IP-Adresse aus und teste diese auf ihre Verfügbarkeit.
- **Übersetzen von Hostnamen in IP-Adressen ohne DNS:** Hier werden DNS-Format Anfragen durch mDNS[34] mittels IP-Multicast über das Netzwerk verschickt. Der mDNSResponder Daemon[35] wertet den Namen in der mDNS-Anfrage aus, und leitet sie an den entsprechenden Dienst direkt weiter. Daraufhin antwortet der Dienst mit seiner Adresse.
- **Automatisches Finden von Diensten in einem Netzwerk:** Eine mDNS-Anfrage für einen bestimmten Dienst wird verschickt, und die Antworten werden in einer Liste von gefunden Diensten gespeichert. Um den Netzwerkverkehr gering zu halten, nutzt Bonjour verschiedene Methoden, wie z.B. das Caching[36] der mDNS-Anfragen.

[31] früher: Rendezvous; entwickelt von Apple
[32] Zero Configuration Networking
[33] Teil der Internet Engineering Task Force (IETF)
[34] Multicast DNS
[35] Hintergrund-Prozess, der Anfragen an bestimmte Dienste verarbeitet
[36] etwas in den Zwischenspeicher aufnehmen

2.3.7.1. Namenskonventionen für Dienste

Die Namenskonventionen der Dienste, die mittels Bonjour publiziert werden, halten sich an die Konventionen, die für DNS gelten. Wie man in Abbildung 2.12 sehen kann, wird der Name vom allgemeinsten Teil, dem . „dot“ (Root-Domain) über die Toplevel-Domain bis hin zum www gesplittet.

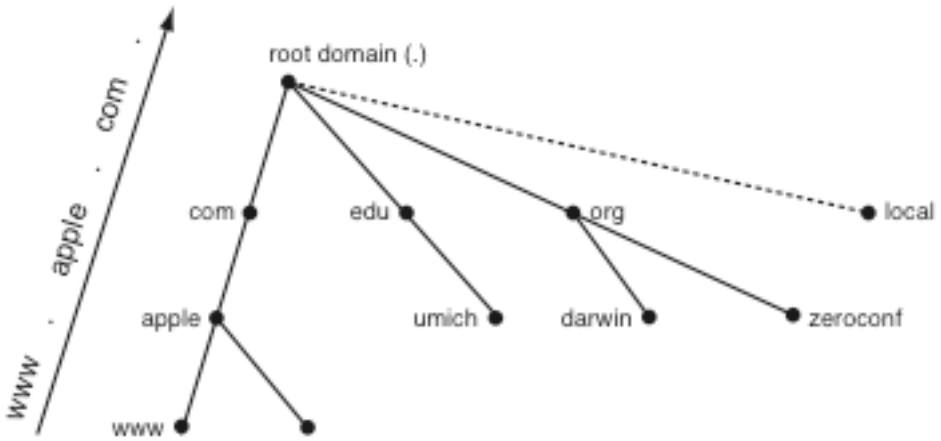

Abbildung 2.12.: Unterteilung von Domainnamen; Quelle: [4, S. 17]

Die Namen der Dienste, die für Bonjour gelten sollen, bilden sich aus der Art des Dienstes sowie dem Transportprotokoll[37] einen Registrierungstyp. Dieser Typ wird genutzt um den Dienst zu registrieren und die mDNS-Anfrage zu bilden.

_SERVICETYPE._TRANSPORTPROTOCOL.

Die IANA[38] verwaltet eine Liste von offiziellen Servicetypen, unter anderem ftp, http oder printer. Ein ftp-Dienst hätte beispielsweise folgenden Registrierungstyp.

_ftp._tcp

Bei der Verwendung eines Namens für die Serviceinstanz wird vorgesehen einen für Menschen lesbaren String zu verwenden. Insgesamt setzt sich der Servicename aus dem Instanznamen, dem Typ, dem Kommunikationsprotokoll,

[37] UDP oder TCP

[38] Internet Assigned Numbers Authority - verantwortlich für die Koordination von DNS Root, IP-Adressierung etc.

der Pseudodomain „local“ und der Rootdomain zusammen (Vgl. Beispiel aus Abbildung 2.13)[4, S. 7-20].

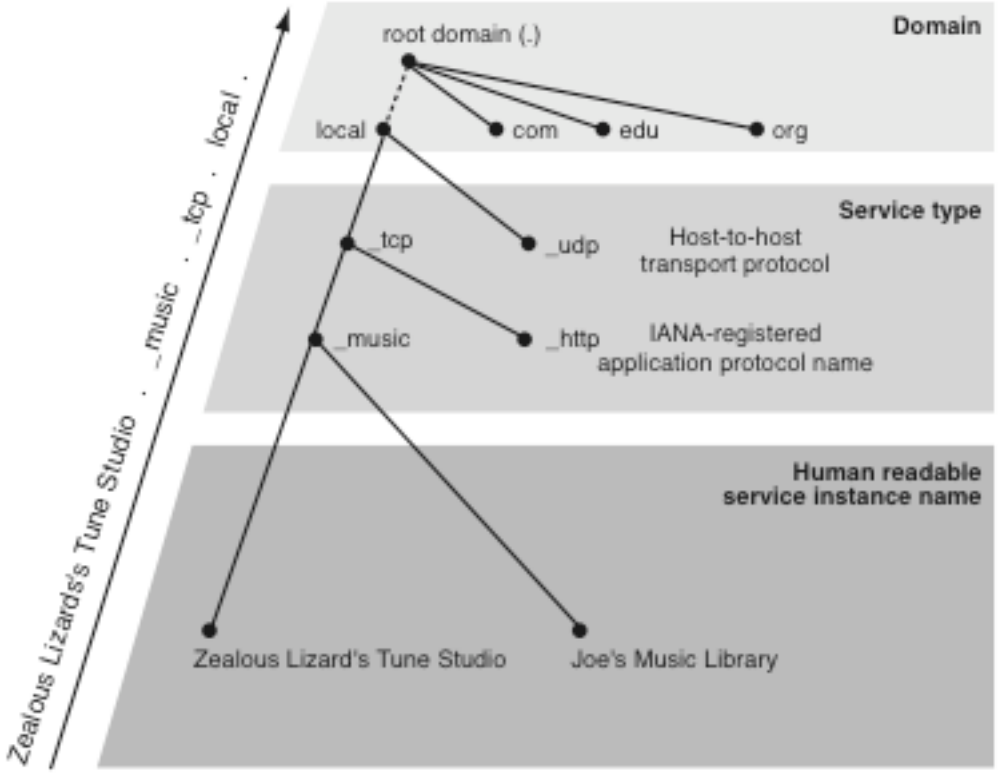

Abbildung 2.13.: Aufbau eines Dienstnamen in Bonjour; Quelle: [4, S. 20]

2.4. OpenGL|ES

OpenGL|ES steht als plattformübergreifende Low-Level Grafik-Bibliothek da und verfolgt den Ansatz eine 3D Welt zu verwalten und Objekte zu verfolgen, welche in diese Welt gesetzt werden. Auf diese Art und Weise wird dem Nutzer ein Fenster in die virtuelle Welt präsentiert. OpenGL ist weitaus komplexer als Quartz[39] allerdings bekommt man hier die volle Hardwarebeschleunigung zu spüren. [LaMa09]

OpenGL|ES ist eine Kompaktversion von OpenGL und wurde speziell für mobile Geräte entwickelt. Während Version 1.0 noch durch Software gerendert werden musste, ist es seit Version 1.1 durch die Hardware möglich. Generell wird OpenGL|ES in zwei Generationen unterteilt. Zum Einen in die 1.x Versionen, welche ohne Vertex und Fragmentshadern auskommen müssen und zum Anderen in die 2.x Versionen die mit Shadern arbeiten können. Sie existieren gemeinsam auf den Systemen, da 2.x nicht abwärtskompatibel ist.[PuAaMi+08] Auf dem iPhone kann man beispielsweise mit der Version 1.1 arbeiten.

Koordinatensysteme und Transformationen sind ein wichtiger Bestandteil von OpenGL und jedes Koordinatensystem bezieht sich auf ein anderes Koordinatensystem. Um das zu verdeutlichen, kann ein Raum in einem Haus sein eigens Koordinatensystem haben, aber ist dennoch in Beziehung zum Koordinatensystem des Hauses. Es gibt also immer ein Referenzkoordinatensystem. Das oberste der Hierarchie ist das sogenannte Welt-Koordinatensystem.

Objekte können durch Punkte im 3D-Raum durch x,y,z Koordinaten beschrieben werden und definieren dadurch eine virtuelle Welt. In OpenGL werden Objekte durch geometrische Primitive beschrieben und durch die Art wie diese verbunden werden (Vgl. Abbildung 2.14). Im Gegensatz zu OpenGL|ES, werden in OpenGL auch Quadrate als Primitive behandelt. Ferner gibt es auch keine double-Werte und Funktionsaufrufe, wie *glBegin* und *glEnd* existieren ebenfalls nicht mehr. Objekte mit ihren Oberflächen werden durch Dreiecke ausgedrückt. Diese Dreiecke werden erzeugt, indem ein Array von Punkten (Vertexe) definiert wird.

[39] weitere Grafikbibliothek auf dem iPhone

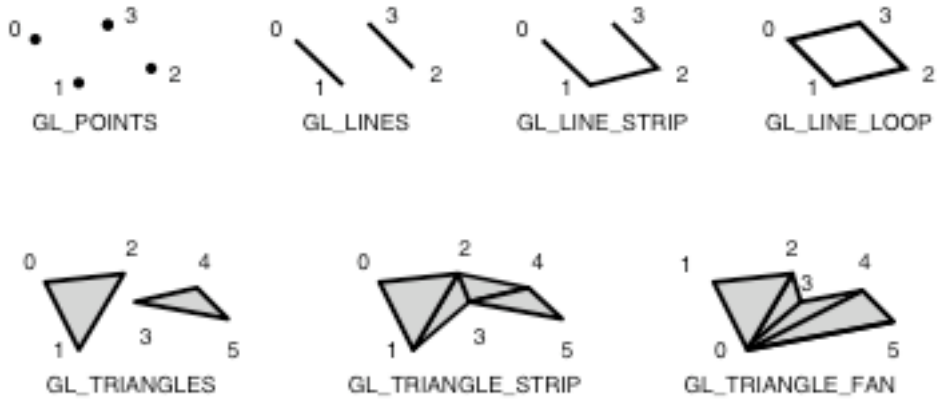

Abbildung 2.14.: geometrische Primitive mit implizierter Verbindung; Quelle: [PuAaMi+08, S. 58]

Ferner hat man die Option Vertexe mit Farbwerte zu definieren oder Oberflächen mit bestimmten Materialeigenschaften und deren Normale für Lichtberechnungen zu belegen. Farbwerte werden durch das RGB-Modell mit Alpha-Erweiterung ausgedrückt. Die Menge an Blau, Rot und Grün sowie der Alphawert im Wertebereich von 0 bis 1 wird somit bestimmt. Da OpenGL eine Zustandsmaschine ist, bleibt ein Farbwert solange erhalten bis dieser geändert wird. Weiterhin unterscheidet OpenGL auch die Art der Lichtquelle nach Punktlicht, gerichtetem Licht und Spotlicht (Vgl. Abbildung 2.15). Auch die Art der Reflexionen können in OpenGL definiert werden. So können Aussagen über ambiente, diffuse und spiegelnde Reflexionen getroffen werden.

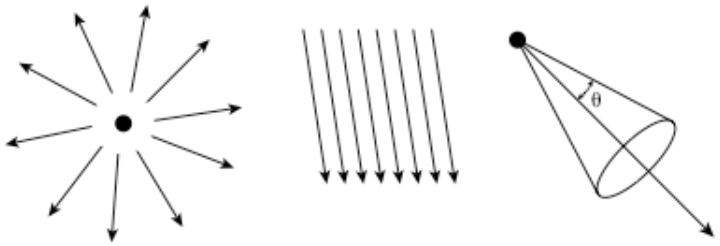

Abbildung 2.15.: Lichtquellen in OpenGL; Quelle: [PuAaMi+08, S. 89]

Um Oberflächen realistisch wirken zu lassen, können diese mit Texturen belegt werden. Dabei kann jedes Dreieck (Polygon) seine eigene Textur erhalten. Als Quelle kann dafür jedes digitale Bild dienen. Die Verankerung der Textur

geschieht an den Polygonecken. Sofern man eine große Fläche mit einer Textur belegen will, kann dies auch mit Wiederholung der Textur erreicht werden, um Speicherplatz im Texturspeicher zu sparen.

Weiterhin bietet OpenGL Methoden zur Reduzierung des Aufwandes einer Szene und um ein korrektes Bild hinsichtlich der Geometrie von Objekten anzufertigen. Eine Technik ist z.B. das Backface Culling, wo anhand der Oberflächennormale die Rückseite der Polygone ermittelt wird, um diese nicht zeichnen zu müssen, da sie ohnehin nicht sichtbar sind. Außerdem werden Polygone, die sich außerhalb des Sichtvolumens befinden, gekappt. Ferner wird auch ein Z-Buffer zum Speichern der Tiefeninformation eines jeden Pixels des Polygons verwendet.

Wie anfangs erwähnt sind Transformationen ein wichtiges Element in OpenGL. Demgemäß sind durch homogene Koordinaten und der Multiplikation mit Matrizen Transformationen von Polygonen leicht zu erreichen. Mögliche Transformationen auf Polygone sind: Rotation, Translation, Scherung, Stauchung und Skalierung (Vgl. Abbildung 2.16). Diese Transformationen sind sogar durch Nutzung eines Matrixstacks geschachtelt möglich, was aufgrund der Implementierung von OpenGL als Zustandsmaschine von erheblichen Vorteil ist. Andernfalls müsste man nach jeder Transformation auf ein Polygon die negative Transformation ausführen um den Ausgangszustand wiederherzustellen. Weiterhin werden Transformationen verwendet um Objektkoordinaten in

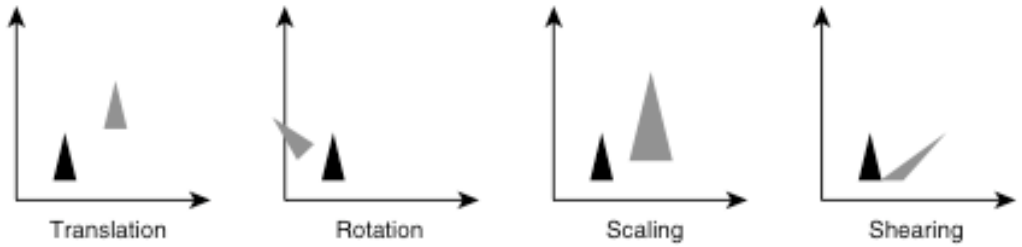

Abbildung 2.16.: affine Transformationen; Quelle: [PuAaMi+08, S. 36]

Kamerakoordinaten und danach auf die Projektionsfläche umzurechnen. Abbildung 2.17 zeigt den Weg eines Objektes bis zum Framebuffer und welche Transformationen nötig sind, um als richtige Darstellung auf den Bildschirm zu landen.

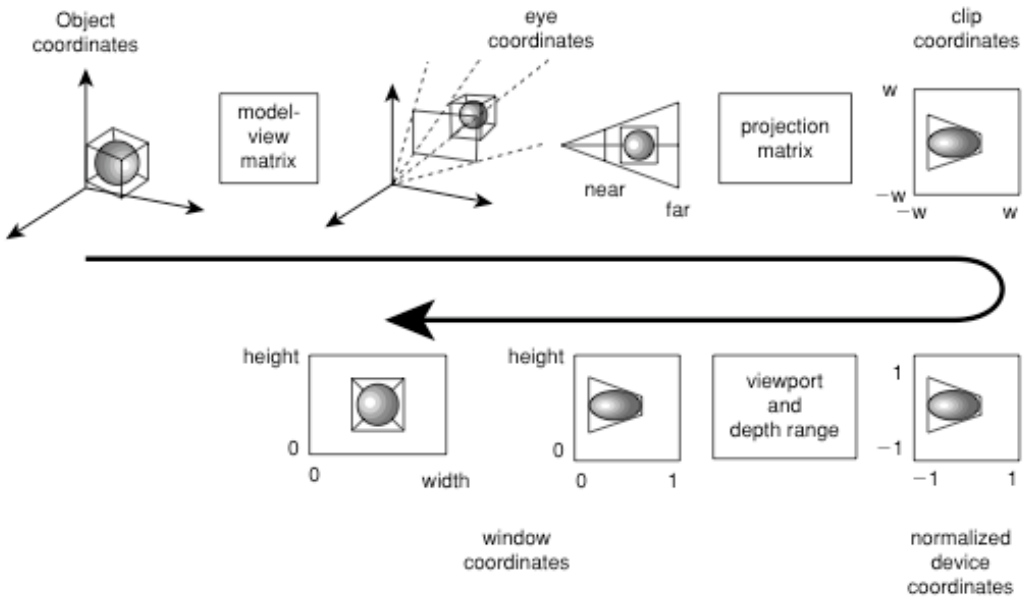

Abbildung 2.17.: Der Weg eines Objektes bis zum Framebuffer; Quelle: [PuAaMi+08, S. 28]

2.5. Programmiersprachen

Es gibt eine Reihe von objektorientierten Programmiersprachen, die zur Entwicklung von Anwendungen auf mobilen Geräten genutzt werden. Je nach Zielplattform kommen nur bestimmte Programmiersprachen in Frage (Vgl. Kapitel 2.1 S. 3). Im Folgenden wird exemplarisch auf Objective-C als Programmiersprache zur Entwicklung für die iPhone Plattform eingegangen. Bei der Sprache handelt es sich in erster Linie um eine objektorientierte Erweiterung der Sprache C. Die folgende Ausarbeitung soll lediglich einen kurzen Einblick in die Programmiersprache geben ohne genauer auf die Konzepte einer objektorientierten Programmierung einzugehen.

2.5.1. Geschichte von Objective-C

In den frühen 80er Jahren wurde meistens mit der strukturierten Programmierung[40] Software entwickelt. Dies führte aber zu Problemen, da mehr und

[40]klassisches Beispiel ist die Programmiersprache Pascal

mehr Prozeduren[41] immer komplexere Kontrollstrukturen geschaffen haben. Somit galt *Smalltalk*[42], die erste rein objektorientierte Programmiersprache, als Lösung des Problems. Selbst einfache Datentypen wie Integer, werden bei Smalltalk als ein Objekt behandelt. [16]

Die Gründer *Brad Cox* und *Tom Love* der Firma *StepStone*, beide mit *Smalltalk* vertraut, entwickelten Anfang 1980 OOPC[43] eine objektorientierte Erweiterung der Programmiersprache C mit Smalltalkelementen.

> „Evolution not Revolution.“ - Brad J. Cox [CoNo91, S. viii]

So beschreibt Brad J. Cox das Prinzip neue Technologien auf altbewährte aufzusetzen. Auf diese Weise war es möglich mit normalen C-Compilern die Programme zu kompilieren und sogar C-Quellcode mit Objective-C Code zu mischen. [CoNo91, S. 58]

Als Steve Jobs die Firma *NeXT* gründete wurden die Rechte von *StepStone* an Objective-C gekauft. NeXT entwickelte auf dieser Basis dann das *NeXTstep* User Interface und den *Interface Builder*, welcher mit der Verwendung objektorientierter Konzepte anderen Systemen dieser Zeit weit voraus war. 1996 wurde dann *NeXT* von *Apple Computer Inc.* übernommen, um Objective-C als Basis für das neue *Mac OS X* zu etablieren. Dies beinhaltete auch den *Project Builder*[44] und den *Interface Builder*.[17]

2.5.2. Syntax und Semantik

Wie bereits erwähnt basiert Objective-C auf der Sprache C. Da C als prozedurale Sprache keine objektorientierten Konzepte kennt, wurden diese durch folgende, meist aufbauend auf Smalltalk, erweitert:

- Objekte und Klassen
- Polymorphismus und Verkapselung
- Vererbung

[41] Prozeduren besitzen im Gegensatz zu Methoden keinen Rückgabewert
[42] weitere Informationen zu Smalltalk: http://www.smalltalk.org/smalltalk/whatissmalltalk.html
[43] Object-Oriented Programming in C
[44] später: Xcode

- Methoden und Messages
- Protokolle
- Kategorien

Das bedeutet das Objective-C den Syntax der Sprache C um objektorientierte Smalltalkelemente vergrößert hat. Die Verwendung von eckigen Klammern im Syntax ist solch ein Beispiel.

Objekte und Klassen

Da in Objective-C auch weiterhin das Prinzip der *Header*-Dateien besteht, werden die Deklarationen in „.h“- und die Definitionen in „.m“-Dateien geschrieben.

Listing 2.1: Deklaration der Player Klasse in Player.h

```
1 @interface Player{
2   //Deklaration der Variablen
3   int points;
4 }
5
6 //Deklaration der Methoden
7 -(id) init;
8 -(void) draw;
9 @end
```

Listing 2.2: Definition der Player Klasse in Player.m

```
1  #import "Player.h"
2  @implementation Player{
3    //Deklaration der Instanzvariablen
4    int points;
5  }
6
7  //Definition der Methoden
8  -(id) init{
9    //initialisierung der Klasse
10 }
11 -(void) draw{
12   //Zeichencode
13 }
14 @end
```

Standardmäßig werden alle Instanzvariablen als *@protected* deklariert. Weiterhin gibt es die Möglichkeit Variablen als *@public* oder *@private* zu deklarieren. Bei Aufruf solch einer Direktive werden alle nachfolgende Variablen dieser

zugeordnet. Durch das Importieren der Headerfile kann die Deklaration der Instanzvariablen in der „.m“-Datei auch weggelassen werden. Dadurch, dass Klassen keinen Konstruktor haben, müssen sie durch Methoden der Klasse initialisiert werden.

Listing 2.3: Erzeugung einer Objektinstanz mit Initialisierung

```
1 Player *player = [[Player alloc] init];
```

Der Aufruf von *alloc* reserviert den Speicher und *init*, eine Methode der Klasse, initialisiert die Instanzvariablen. [6, S. 35-47]

Vererbung

Wie auch bei vielen anderen objektorientierten Sprachen, gibt es in Ojective-C das Konzept der Vererbung. Das Prinzip der Mehrfachvererbung wird nicht unterstützt, kann aber durch Verwendung von Protokollen (Vgl. Protokolle S. 38) realisiert werden. [3]

Listing 2.4: Das Ableiten einer Klasse

```
1 @interface Player : SUPERCLASS{
2 }
3 @end
```

Properties

Properties stellen einen einfachen Weg in Objective-C dar, um Setter- und Getter Methoden zu deklarieren und implementieren. Anstatt für jede Instanzvariable eine dieser Methoden zu schreiben, ist es ausreichend diese im Interface-Teil zu deklarieren und im Implementations-Teil synthetisch zu bilden. Das bedeutet, dass beim Kompilieren automatisch Setter und Getter erstellt werden. Ein weiterer Vorteil durch Properties ist die Möglichkeit der Verwendung des Dot-Syntaxes. [6, S.57-59]

Listing 2.5: Erstellen von Setter und Getter Methoden durch Properties

```
@interface Player{
  int x;
}
@property int x;
@end

@implematation Player{
@synthesize x;
}
@end
//Dot-Syntax durch Properties
//beide sind valid
[player setX:2];
player.x = 2;
```

Methoden und Messages

Der Aufruf einer Methode eines Objektes geschieht durch Messages[45]. Nach dem Senden der Nachricht erfolgt die Suche nach der entsprechenden Methode zur Laufzeit im Objekt.

Listing 2.6: Message an ein Objekt

```
[player draw];
```

In dem Beispiel 2.6 wird lediglich der Name der Methode an das Objekt gesendet, was schließlich die Ausführung der Methode zur Folge hat. Im folgenden Listing 2.7, erkennt man einen Methodenaufruf mit Parametern, im üblichen C-Style sowie in Objective-C. Beide Varianten sind möglich. Bei Objective-C kann der Methodenname durch Doppelpunkte getrennt sein. Jeder Doppelpunkt ist Platzhalter für einen Methodenparameter. Diese Art des Syntax bietet den Vorteil, Methoden und deren Parameter, die bei einem Aufruf benötigt werden, genauer zu beschreiben.

Listing 2.7: Methodenaufruf mit Parametern

```
//C
initPlayerAtPosition( 2, 3, 4);

//Objective-C
[player initPlayerAtPositionX: 2 positionY: 3 positionZ: 4];
```

[45] dt.: Nachrichten

Weiterhin bietet Objective-C die Option Messages mit einem *nil*-Wert zu verschicken. Dies hat zwar keine Auswirkungen zur Laufzeit, aber einige Abläufe in Cocoa gewinnen daraus einen Vorteil. [6, S. 15-20]

Kategorien

Anhand von Kategorien ist es möglich bereits bestehende Klassen um Funktionalitäten zu erweitern ohne den Quellcode dieser zu kennen. Die Deklaration einer Kategorie unterscheidet sich nur geringfügig von der einer Klasse. Die Methoden dieser Kategorie werden beim Kompilieren zu der bestehenden Klasse hinzugefügt und haben auch Zugriff zu allen Instanzvariablen der ursprünglichen Klasse. Kategorien kann man als Alternative zur Vererbung sehen. Häufige Nutzung erhalten Kategorien dadurch, dass mit ihnen eine Kapselung von Methoden zu erreichen ist, da die Direktiven public, private und protected nur für Instanzvariablen gelten. [6, S. 69-72]

Listing 2.8: Deklaration und Definition einer Kategorie

```
#import "Player.h"
@interface Player (KATEGORIENAME){
  //Deklaration der Methoden
}
@end

@implementation Player (KATEGORIENAME) {
  //Definition der Methoden
}
@end
```

Protokolle

Protokolle deklarieren Methoden die von jeder Klasse benutzt werden können. Die Klassen, welche diese Protokolle adoptieren, müssen eine eigene Implementierung dafür bereitstellen. Das Pendant zu Protokollen bilden die Interfaces in Java. [6, S. 73-74]

Listing 2.9: Nutzung eines Protokolls

```
@interface Player <PROTOKOLLNAME>{
}
@end
```

2.6. Zusammenfassung

Im Vorherigen wurde schon mehrmals Bezug auf das iPhone genommen. Dies hat den Hintergrund, dass das iPhone zur Umsetzung der Arbeit in der Lage zu sein scheint. Im Folgenden wird ein ausgiebiger Vergleich zwischen den Geräten stattfinden, um sich für den weiteren Verlauf auf eines festzulegen, da unterschiedliche Techniken gegebenenfalls andere Implementierungen erfordern.

Um mein Spiel zu realisieren, werden folgende Sensoren benötigt: Zum Einen muss es die Möglichkeit geben sich lokalisieren zu lassen, z.B. mittels eines GPS-Empfängers. Zum Anderen wird ein Schwerkraft/Beschleunigungssensor benötigt, der für die Bedienung erforderlich ist. Des weiteren ist die Nutzung von WLAN's und 2G/3G Netzwerken erforderlich und das Gerät muss in der Lage sein, grafiklastige Anwendungen auszuführen. Nahezu alle der favorisierten Smartphones erfüllen diese Ansprüche, bis auf das Blackberry Storm, da es keine Möglichkeit bietet ein WLAN zu nutzen. Auch dem Palm Pre mangelt es an Grundvoraussetzungen, da es für grafiklastige Anwendungen nicht ausgelegt ist. Auf das N97 kann auch verzichtet werden, da es zu Beginn dieser Arbeit noch nicht erhältlich war.

Generell ist die Implementierung des Spiels für die restlichen Geräte denkbar. Aber auch hier gibt es gewisse Fälle, die beachtet werden müssen. Solch ein Fall ist die Gerätebasis. Dies ist vergleichbar mit der Programmierung für eine Spielkonsole wie die Playstation. Die Spielerfahrung ist weltweit auf jeder Playstation die gleiche. Genauer gesagt, kann es vorkommen, dass wenn man für Android oder Symbian programmiert, gewisse Sensoren auf einem Smartphone nicht zur Verfügung stehen können oder starke Unterschiede in der Leistung einzelner Geräte vorhanden sind. So kann beispielsweise der Beschleunigungssensor auf unterschiedlichen Geräte andere Werte erzeugen, wodurch die Steuerung der Spielfigur sehr verschieden sein kann. Dadurch ist die Ausführung des Spiels auf einigen Geräten nicht möglich oder es gibt unterschiedliche Spielerlebnisse, die im 2 Spielermodus in Ungerechtigkeit enden können. Das bedeutet für den Programmierer, dass er alle Eventualitäten abdecken muss. Beim iPhone 3G und iPhone 2G dagegen kann er sich darauf verlassen, dass alles angesprochen werden kann und die Performance und Ausführung immer

die gleiche ist. Folglich spricht das Argument der Gerätebasis für das iPhone. Weiterhin kann anhand von Benchmarks [10] eine sehr gute Performance im grafiklastigen Bereich für das iPhone gegenüber anderen Telefonen erkannt werden. Ferner bietet das iPhone die Unterstützung von Bonjour an, welches ein einfaches Finden von Netzwerkdiensten ermöglicht. Aufgrund der Netzwerkfähigkeit des zu entwickelnden Spiels, ist dies ein enormer Vorteil.

Aus der Gegenüberstellung der Mobiltelefone geht hervor, dass neben der persönlichen Neigung und aufgrund der zuvor erläuterten Vorteile, das iPhone 3G als Basis zur Verwirklichung der Projektarbeit am besten geeignet ist.

Abbildung 2.18.: Das iPhone 3G; Quelle: [26]

3. Analyse

In diesem Kapitel werden die Spiele, die auf Downloadplattformen angeboten werden, genauer nach gewissen Kriterien, wie Spielspaß, Funktionsumfang, Bedienung und grafische Benutzeroberfläche untersucht, um eine allgemeine Tendenz, in Hinblick auf die Erstellung des Konzeptes für das eigene Spiel, festzustellen. Aus dem Konzept werden dann die resultierenden Anforderungen an das System ermittelt und Anwendungsfälle aufgezeigt. Anschließend werden Ansätze für den Systementwurf erläutert.

3.1. Analyse der Spiele auf Downloadplattformen

Ein Grund für den Erfolg des iPhones ist sicherlich der App Store[46], in welchem bekannte Firmen wie z.B. Electronic Arts sowie selbständige Entwickler ihre Produkte veröffentlichen können. Somit wurde das iPhone ein ernster Kontrahent zu den mobilen Spielgeräten, wie die Sony PSP[47]. Neben dem App Store existieren noch weitere Downloadplattformen. So hat Google den Android Market und Nokia den OVI Store eingerichtet. Da aber diese beiden Plattformen, im Vergleich zum App Store, ein sehr spärliches Angebot an Spielen besitzen, werden vorzugsweise Spiele, die im App Store angeboten werden, betrachtet.

Von den zurzeit 56.000 angebotenen Programmen sind in etwa 10.000 als Spiele zu identifizieren. Hinzu kommt, dass zur Zeit ca. 170 Spiele pro Monat ihren Weg in den App Store finden [1]. Im Kern wird das zu entwickelnde Spiel Ähnlichkeiten oder eine Mischung aus im weitesten Sinne Autorennspiele und Geschicklichkeitsspiele darstellen. Spiele, die diese Genre repräsentieren sind

[46] Apple's Verkaufsplattform von Software fürs iPhone
[47] Playstation Portable

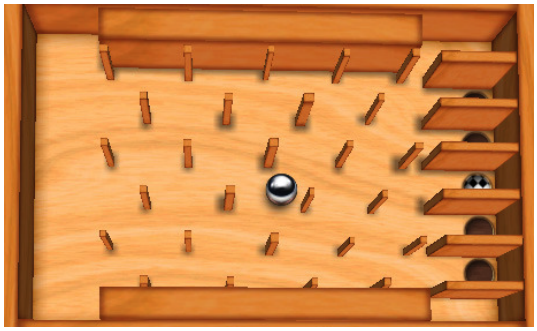

Abbildung 3.1.: Wooden Labyrinth 3D, Quelle: App Store

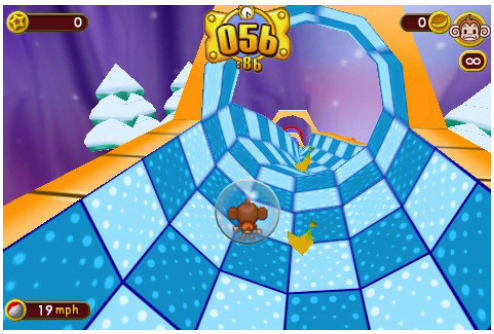

Abbildung 3.2.: Super Monkey Ball, Quelle: App Store

Wooden Labyrinth 3D und *Super Monkey Ball* (Vgl. Abbildung 3.1 und 3.2). Ähnlichkeiten sind aufgrund der feinen Steuerung beim Labyrinthspiel und das Ausweichen von Objekten bei den Rennspielen auszumachen. Anhand der Rezessionen zu den Spielen wird erkennbar, dass eine schöne Grafik und saubere Steuerung als wichtig anzusehen sind.

Wenn man die Vielzahl der Spiele, die im App Store angeboten werden untereinander vergleicht, können viele Eigenschaften für erfolgreiche Spiele ermittelt werden.

Anwendung:

Im Allgemeinen ist zu sagen, dass die Anwendungen meist als Spiele für Zwischendurch konzipiert sind. Sie verfolgen oft den Ansatz den Nutzer schnell ins Spiel zu bringen. Weiterhin sind Levels, Missionen, Rätsel etc. für eine Dauer von nur wenigen Minuten entworfen. Im Gegensatz zu Anwendungen auf Heim-PC's, wo meist ein langes Laden, Intro und Konfiguration von Grafik und Sound vor dem Start ins Spiel mit einhergeht und die Zeit, die für die Erfüllung von Aufgaben benötigt wird, weitaus höher ist.

Graphische Benutzeroberfläche und Bedienung:

Durch die begrenzte Auflösung des Bildschirms halten viele Spiele ihre GUI[48] absichtlich einfach. Das bedeutet, dass bewusst auf einige Elemente verzichtet wird, um die Darstellung des eigentlichen Spielgeschehen in den Vordergrund zu rücken. Ähnlich verhält es sich mit der Bedienung, die bei den meisten Spielen sehr intuitiv ist. Generell kann man zwei Arten der Bedienung feststellen. Zum Einen die Nutzung des Beschleunigungssensors sowie die Implementierung eines virtuellen Steuerkreuzes auf dem Display, sofern dieser Multitouch fähig ist und zum Anderen wird bei Mobiltelefonen mit physischer Tastatur im Allgemeinen die Steuerung durch Tastendruck realisiert. Grenzen bei Bedienung und Darstellung sind z.B. bei der portierten Version von Sim City[49] zu

Abbildung 3.3.: Sim City; Quelle: App Store

erkennen. Es bereitet keine Freude mit der beschränkten Eingabemöglichkeit und dem kleinen Ausschnitt des Spielgeschehens ein an sich komplexes und vielfältiges Spiel zu spielen, was auch durch folgendes Zitat einer Kundenrezession belegt wird:

> „Zu aufwendig ist es einzelne Elemente auszuwählen und zu positionieren“

[48] Graphical User Interface
[49] eines der erfolgreichsten Wirtschaftssimulationen auf dem PC

Nutzung der Sensoren:

Eine weitere Auffälligkeit ist, dass die Mehrzahl der Spiele die integrierten Sensoren der Mobiltelefone voll ausnutzten. Grundsätzlich kann gesagt werden, dass ein Spiel mindestens auf einen Sensor zugreift.

Mehrspielerfähigkeit:

Die Möglichkeit gegen Freunde zu spielen, kann nur bei einem kleinen Teil der angebotenen Spiele entdeckt werden. Hervorgerufen wird dies durch entweder fehlende Implementierung oder es besteht keine Notwendigkeit aus dem Spielkonzept heraus. Nur sehr wenige Spiele haben einen Realtime-Multiplayer Modus. Eines der ersten dieser Art ist das Autorennspiel Asphalt 4 (Vgl. Abbildung 3.4) auf dem iPhone von der Firma Gameloft. Hier ist es möglich in einem WLAN gegen Freunde anzutreten.

Abbildung 3.4.: Das Spiel Asphalt 4; Quelle: App Store

Spiele die über das Internet gespielt werden, beschränken sich vorwiegend auf sogenannte Browsergames.

Spielerbindung:

Eine Spielerbindung kann durch einen Highscore erreicht werden. Die meisten Spiele beinhalten eine Highscoreliste, die je nach Spielprinzip in Punkte, Anzahl der Züge oder verstrichene Zeit bis zum Erfolg der Aufgabe sortiert ist. Dabei sehen die Implementierungen auch hier sehr unterschiedlich aus. Einige bieten eine lokale persönliche Highscoreliste an, die nur für diesen Spieler gilt, während die Umsetzung bei anderen mit Zugriff auf das Internet weltweit realisierbar ist.

3.2. Das Spielkonzept

Die allgemeine Spielidee ist relativ einfach zu beschreiben. Ziel des Spieles ist es so viele Punkte wie möglich zu erreichen, sei es im 1-Spieler oder Mehrspieler-Modus. Das Spiel gegen einen Mitstreiter ist seit jeher ein Mittel, um den Spaß am Spiel zu vergrößern. Als Genre kann dieses Spiel als Geschicklichkeitsspiel interpretiert werden, da es auf ein schnelles Reaktionsvermögen ankommt.

Der Spieler steuert seine Spielfigur, die sich in einer Art Gang befindet - im folgenden als Spielfeld bezeichnet - und kann sich nur in einem gewissen Bereich des Spielfeldes bewegen. Der Blickwinkel auf die Spielfigur ist dabei aus der Zentralperspektive zu sehen, so dass sich die Spielfigur leicht vor einem befindet. Des weiteren kommen aus der Ferne des Spielfeldes unterschiedliche Objekte auf die Spielfigur zu. Diese können entweder bösartig oder wohlwollend sein. Die Aufgabe des Spielers ist es nun, diese Objekte, sofern sie wohlwollend sind, einzusammeln oder anderenfalls zu vermeiden. Trifft der Spieler auf ein bösartiges Objekt so verliert er eines von fünf Leben. Die anderen Objekte variieren in ihren Eigenschaften. So gibt es am häufigsten weiße Bälle, die es gilt einzusammeln, da sie einen Punkt bringen. Ein anderes Spielobjekt gibt dem Spieler fünf Punkte, was aber gleichzeitig mit der Zunahme des Radius der Spielfigur einhergeht. Dies kann im 2 Spieler Modus von Vorteil sein, da man so eher Punkte sammeln kann, aber eventuell auch eher getroffen wird. Weiterhin kann man gelbe Würfel einsammeln, die viele Überraschungen bereit halten. So können durch das Einsammeln die bösartigen Objekte an Geschwindigkeit zunehmen oder abnehmen. Außerdem ist ein Rücksetzten des Spielerradius möglich sowie die Zugewinnung eines Extra-Lebens, oder der Gewinn der Unsterblichkeit für 15 Sekunden. Darüber hinaus ist das Spiel in Level unterteilt.

So startet man im ersten Level mit sich langsam bewegenden Objekten, die je mehr Punkte man sammelt, schneller werden und in der Anzahl wachsen. Wenn die Anzahl der Leben 0 erreicht hat, ist man tot und hat die Option sich in der Online-Highscoreliste, entweder mit oder ohne Positionsdaten zu verewigen.

Im Mehrspieler-Modus gelten die selben Regeln nur das der Gewinner jener ist, der länger überlebt, unabhängig von der gesammelten Anzahl der Punkte. Jedoch kann sich der Spieler, der verloren hat, auch hier in die Highscoreliste eintragen, sofern er gut genug war.

3.3. Resultierende Anforderungen

Für die Umsetzung eines kurzweiligen Spiels gelten gewisse Anforderungen, die es zu realisieren gilt.

Die Zeit bis zum Spielbeginn sollte möglichst minimal gehalten werden und ferner muss allein durch das Spielkonzept die Eingewöhnungszeit, um das Spiel spielen zu können, gering sein. Dadurch wird ein Tutorial innerhalb des Spieles überflüssig, da der Spieler schnell weiß welche Aufgabe er zu erfüllen hat. Sofern der Spieler einen Highscore erzielt hat, soll die Möglichkeit angeboten werden sich in die Online-Datenbank über das Internet einzutragen. Für den Eintrag können neben dem Namen und der erreichten Punktzahl, bei Zustimmung auch seine Positionsdaten mit einfließen.

Durch Nutzung einer anspruchsvollen 3D-Grafik soll die Lust am Spiel gesteigert werden. Hierbei ist darauf zu achten, dass die Performance nicht darunter leidet. Um ein flüssiges Spielerlebnis wahrzunehmen soll die Framerate des Spiels nicht unter 20 Bilder pro Sekunde fallen. Aus diesem Grund wird es ratsam sein, die Anzahl der Polygone zur Beschreibung von Spielobjekten und der Spielfigur möglichst gering zu halten. Die Darstellung der Spielfigur als Ball scheint angemessen zu sein.

Die Steuerung der Spielfigur soll mit Hilfe eines Beschleunigungssensors umgesetzt werden, um so den Anspruch eines Geschicklichkeitsspiel zu gewährleisten. Bei der Bedienung der Spielfigur ist darauf zu achten, dass sie in der Lage

ist schnelle Bewegungen zu vollführen, aber dennoch eine feine und millimetergenaue Steuerung zulässt. Ferner besteht eine Anforderung darin, dem Nutzer auch das Verändern seines Blickwinkels oder der Zoom-stufe durch Berührung des Bildschirms zu bieten. Weiterhin soll der Nutzer auf Geschehnisse im Spiel Rückmeldungen in Form von Audioausgaben bekommen.

Da die Verwendung des Spiels zum großen Teil unterwegs stattfinden wird, ist daher die Implementierung einer Pause-Funktion zu erwägen.

Eine besondere Rolle kommt dem Mehrspielermodus zugute. Der Mehrspielermodus wird, aufgrund des Spielkonzepts auf zwei Spieler begrenzt. Eine wichtige Forderung in Bezug auf die Mehrspielerfähigkeit ist die Gewährleistung einer gewissen Fairness. Hier muss sichergestellt werden, dass eine gleichberechtigte Auswertung des Spielgeschehens stattfindet, zumal sich beide Spieler auf einem Gerät sehen können. Dadurch wird es weiterhin erforderlich sein, die Kommunikation zwischen den Endgeräten so zu gestalten, dass auch die Bewegung des zweiten Spielers als flüssig angesehen wird. Außerdem muss das Spiel robust gegen Verbindungsabbrüche sein.

In Anbetracht der eingeschränkten Hardwarevoraussetzungen muss darüber hinaus auf die zur Verfügung stehenden Ressourcen Acht genommen werden. Ferner ist es wichtig das System so modular zu entwerfen, dass eine Erweiterung des Spiels ohne große Probleme möglich ist.

3.4. Use-Case Analyse

Prinzipiell lassen sich aus dem Spielkonzept und der Anforderungsanalyse drei Anwendungsfälle im Spiel feststellen (Vgl. Abbildung 3.5).

Anzeigen des Highscores

Nach dem Start des Spiels hat der Nutzer die Möglichkeit sich die aktuelle Top 10 der besten Spieler anzugucken und gegebenenfalls den Ort, an dem dieser Highscore aufgestellt worden ist, zu begutachten.

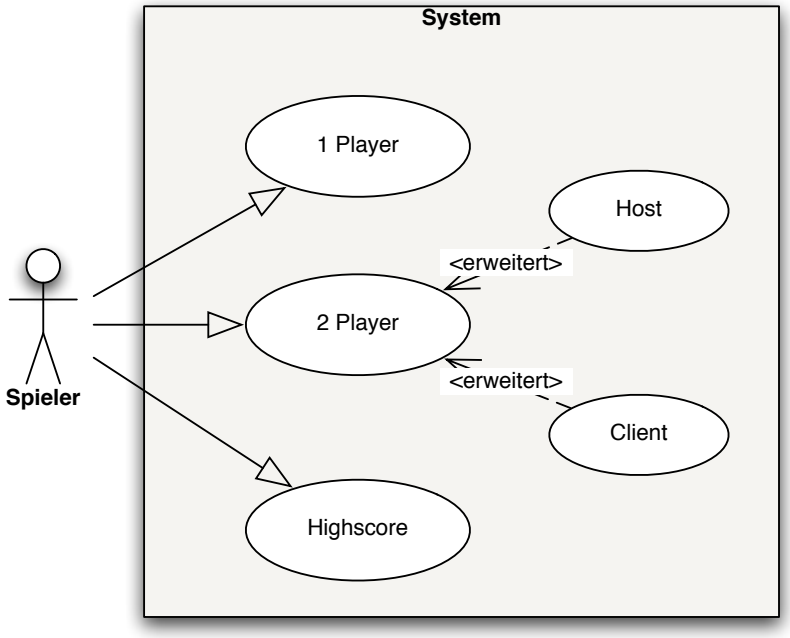

Abbildung 3.5.: Use-Case Diagramm

Starten des Solospiels

Beim Start des Solospiels ist es die Aufgabe des Spielers so viele Punkte wie möglich zu sammeln. Der Gameloop (Vgl. Abbildung 3.6) läuft so lange bis der Spieler seine Leben aufgebraucht hat. Anschließend hat der Spieler die Möglichkeit sich mit seinem Namen, der Punktzahl und gegebenenfalls seiner Position in den Highscore einzutragen.

Starten des 2 Spieler Spiels

Der Ablauf des Mehrspielerspiels ist der gleiche, nur dass der Gameloop so lange läuft bis beide Spieler kein Leben mehr haben. Im Vorfeld hat ein Spieler die Möglichkeit das Spiel zu starten, während der andere es beitreten kann (Vgl. Abbildung 3.7).

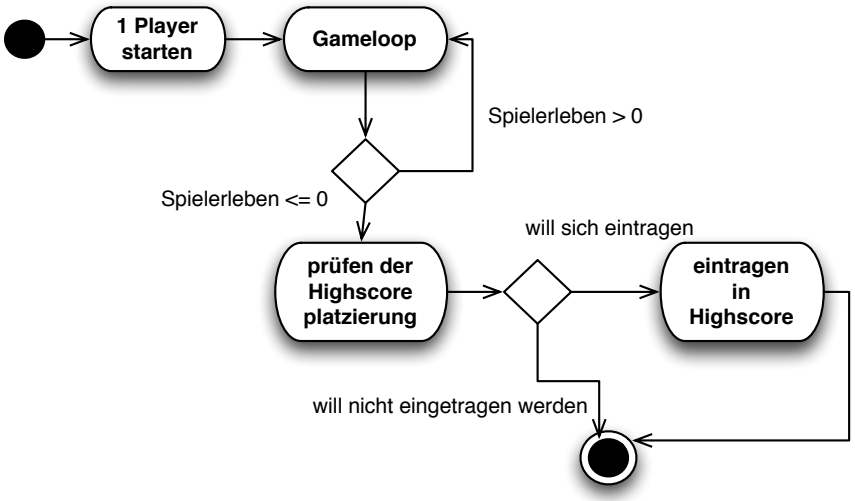

Abbildung 3.6.: Aktivitätsdiagramm des 1 Spieler-Modus

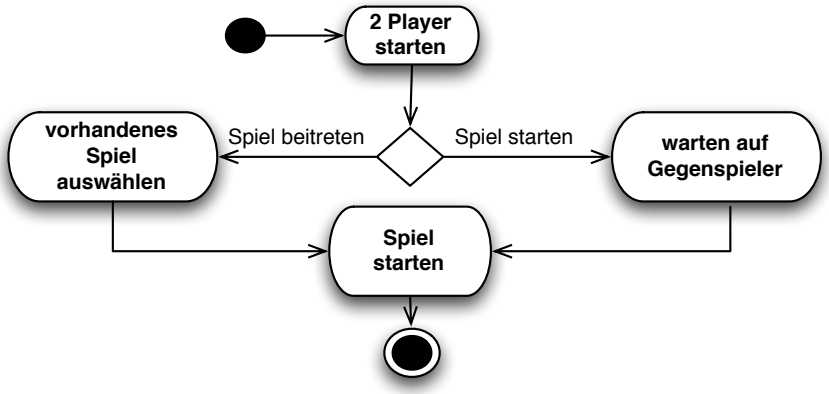

Abbildung 3.7.: Aktivitätsdiagramm des 2 Spieler-Modus

3.5. Ansätze für den Systementwurf

Um mit dem Systementwurf beginnen zu können, müssen im Vorfeld einige wichtige Entscheidungen getroffen werden, die aus den Anforderungen hervorgegangen sind. Zum Einen sollte das Programm nach der MVC-Architektur[50] entworfen werden, um so eine modulare und leicht erweiterbare Software zu schaffen. Das heißt, dass die Software in drei Schichten, nämlich der Datenhaltung, der Anwendungslogik und der Präsentation, unterschieden wird.

[50] Model-View-Controller

Zum Anderen, ist es wichtig zu überlegen wie der Mehrspielermodus zu realisieren ist. Da das iPhone mehrere drahtlose Methoden bietet sich mit einem Netzwerk zu verbinden, müssen auch hier die Vor- und Nachteile der jeweiligen Zugangstechniken gegenübergestellt werden. Für Datenübertragungen kommen EDGE, UMTS und Wi-Fi (Vgl. 2.3.5) in Frage, die sich bezüglich der Übertragungsrate erheblich unterscheiden. Die Übertragungsrate ist wichtig um die Fairness und die Reaktionszeit in einem Realtime-Multiplayer Spiel zu gewährleisten. Durch die Nutzung von Wi-Fi kann von einer hohen Datenrate, je nach Internetverbindung ausgegangen werden. Trotzdem kann hier eine hohe Latenz[51] vorhanden sein, wenn der Spielgegner am anderen Ende der Welt sitzt. Als Latenz wird die Zeit bezeichnet, die das Signal von einem Kommunikationspartner zum anderen benötigt. Wenn die Zeit zwischen t0 und

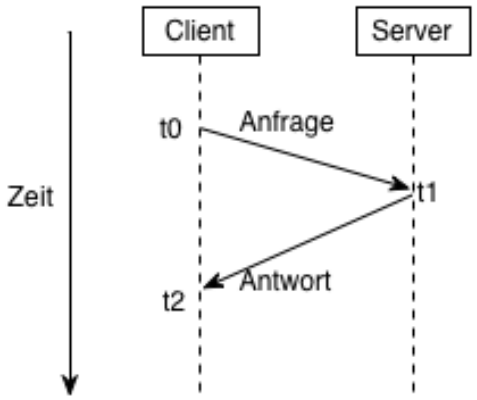

Abbildung 3.8.: Zeitverlauf in der Client-Server Kommunikation

t2 zu hoch (Vgl. Abbildung 3.8) ist, müssen verschiedene Kompensationstechniken zur Verminderung der Latenz angewendet werden. Techniken, um dieses zu realisieren, sind die Vorhersage von Ereignissen bzw. Bewegungen und Zeitmanipulationstechniken[52] [ArClBr08][Mauv00]. Ein weiterer Nachteil bei einem Spiel über das Internet ist, dass noch zusätzlich Infrastruktur in Form eines Servers geschaffen werden muss. Der Server hätte die Aufgabe die Spieler zu verwalten und außerdem eine Kommunikation zwischen den Teilnehmern herzustellen (Vgl. Abbildung 3.9). Aus diesen Gründen wird in dieser Arbeit auf die Implementierung einer zentralen Serverinstanz im Internet verzichtet

[51] als Faustregel: Latenz in ms: Entfernung zum Gegner in Kilometern / 300
[52] Time Warp, Time Delay, Dead Reckoning

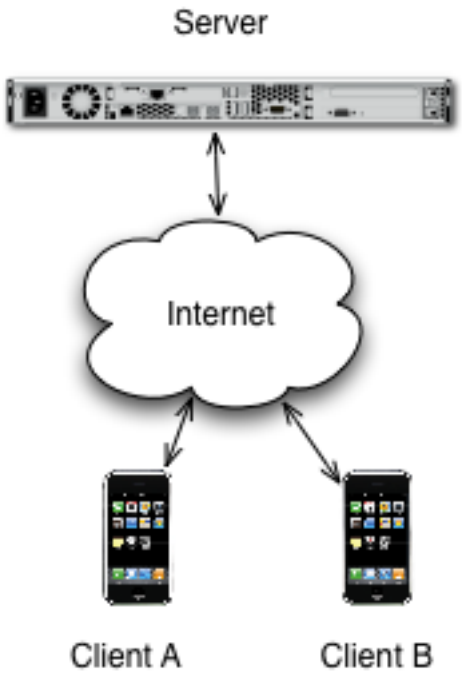

Abbildung 3.9.: Spielaufbau durch Server übers Internet

und in Folge dessen wird sich die Mehrspielerfähigkeit auf das lokale Netzwerk beschränken, da von optimalen Bedingungen, in Anbetracht der Latenz und Übertragungsgeschwindigkeit, ausgegangen werden kann.

Die Kommunikation zwischen den Spielern wird mittels Sockets realisiert werden. Hierbei ist es wichtig zwischen TCP (Stream Sockets) und UDP (Datagram Sockets) zu unterscheiden. Durch den Vergleich der Eigenschaften von UDP und TCP (Vgl. Kapitel 2.3.4 S. 18) wird trotz der geringen Zuverlässigkeit UDP verwendet, da es das Protokoll ist, worauf es ankommt wenn ein Datenaustausch mit geringer Latenz stattfinden soll.

Um weitere Überlegungen zum Aufbau des Systems machen zu können, muss zunächst der Begriff *Gamestate* definiert werden. Der *Gamestate* ist die Zusammenfassung aller wichtigen Informationen, die den Status eines Spiels beschreiben. Im Detail sind das Informationen zu Bewegungen bzw. Positionen von Objekten und statistische Daten, wie z.B. Punktanzahl. Informationen zur Kameraposition eines Spielers sind hingegen für den anderen Teilnehmer eher unwichtig. Der Gamestate ist daher zentrales Element, wenn es um die

Kommunikation bzw. die Synchronisation von Netzwerkspielen geht. Folglich ist die Frage, wie der Gamestate synchronisiert wird und wer die Befugnis hat ihn zu verwalten.

Es gibt zwei Konzepte (Vgl. Client-Server und Peer-to-Peer 2.3.2 S. 12), um dieses zu realisieren. Bei einem Ansatz mit der Peer-to-Peer Architektur[53] gibt es keine Instanz, welche die Kontrolle über den Gamestate hat. Hier hält jeder Teilnehmer seinen eigenen Gamestate und muss dafür sorgen, dass er den anderen Teilnehmern Änderungen am Gamestate übermittelt. Der Empfänger solch einer Änderung muss davon ausgehen, dass die Änderung valide ist. Durch diesen Ablauf und das Weglassen der zentralen Kontrollinstanz bietet sich so die Möglichkeit das Spiel zu cheaten[54] und ferner ist es zudem weitaus schwieriger die Synchronisation zwischen den Teilnehmern herzustellen.

Dadurch ergibt sich, dass wie bei vielen anderen Netzwerkspielen, auf die Client-Server Architektur zurückgegriffen wird, da sie leichter zu implementieren und der Gamestate leichter zu verwalten ist. Der Server fungiert hier als Kontrollinstanz über den Gamestate und gibt Änderungen an die Clients weiter. Diese empfangenen Änderungen können von den Clients durch die serverseitige Validierung als richtig gewertet werden. Dennoch gibt es zwei unterschiedliche Arten wie der Systementwurf aussehen könnte.

Zentralisierte Client-Server Kontrolle: Der Client stellt eine Anfrage an den Server, weil er seinen Gamestate ändern will. Der Server validiert diesen Wunsch der Änderung und antwortet entsprechend mit JA oder NEIN.

Dezentralisierte Client-Server Kontrolle: Der Client teilt dem Server mit, dass er seinen Gamestate geändert hat. Der Server aktualisiert nach der Validierung seinen globalen Gamestate und übermittelt ihn wieder zurück.

Der Vorteil gegenüber der zentralisierten Variante ist, dass der Client nicht auf die Antwort warten muss. Um dieses Konzept in jedem beliebigen Netzwerk umsetzen zu können, bedarf es der Notwendigkeit, dass ein Client auch als Server fungieren kann. In diesem Fall tritt ein Client als *Host* auf und agiert wie ein Server. Obwohl dieses Prinzip oberflächlich nach Peer-To-Peer aussieht, ist es dennoch eine Client-Server Architektur.

[53] das Onlinespiel Diablo ist ein typischer Vertreter
[54] dt.: schummeln

Als Host wird der Spieler bezeichnet, der ein Spiel startet während der Spieler, der dem Spiel beitritt, als Client tituliert wird.

4. Systementwurf

Das zu entwickelnde System soll laut der Anforderungen ein kurzweiliges 2 Spieler Netzwerkspiel werden. Aus der Diskussion zum Systementwurf ging hervor, dass es sich hierbei, sofern es im Mehrspielermodus gespielt wird, um eine Client-Server Anwendung handeln wird. Aus Abbildung 4.1 kann der grobe Systementwurf entnommen werden.

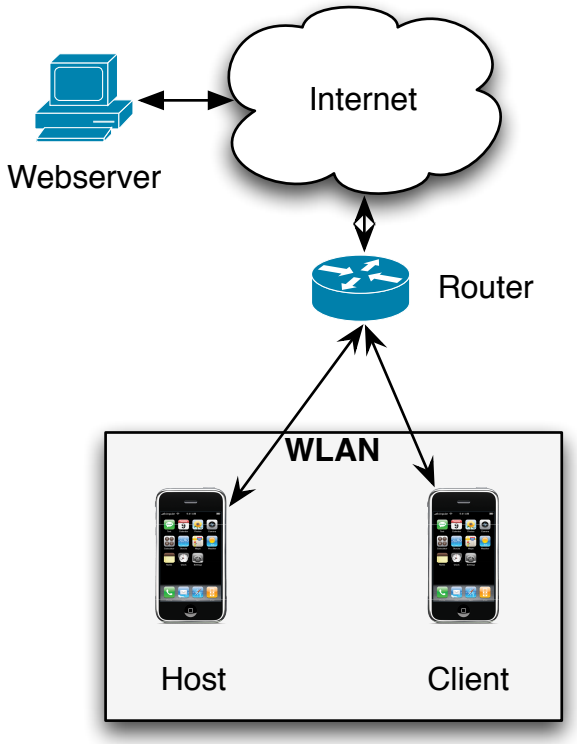

Abbildung 4.1.: grobe Systemarchitektur

Im folgenden werden vermehrt UML-Diagramme[55] verwendet um eine bestimmte Sicht auf Verhalten und Aufbau des Systems ersichtlicher darzustellen.

[55]Unified Modeling Language

4.1. Systemarchitektur

Die Anwendung muss als verteilte Architektur realisiert werden, da eine Anforderung darin besteht eine weltweit zugängliche Highscoreliste bereitzustellen. Ferner müssen durch das Weglassen eines Spielservers, die Clients so konzipiert werden, dass diese auch als Server (Host) fungieren können (Vgl. Kapitel 3.5 S. 53). Somit sind letztendlich keine großen Unterschiede im System zwischen den Anwendungsfällen *1-Player* und *2-Player* gegeben. Dieses Entwurfsmuster für Clients, lässt sich durch den Begriff eines *Fat-Clients* beschreiben. Ein Fat-Client zeichnet sich dadurch aus, dass die Verarbeitung von Daten auch auf den Client ausgeführt wird, im Gegensatz zum *Thin-Client* der lediglich die Benutzerschnittstelle zum Server darstellt. Durch diesen Ansatz kommunizieren Client und Host nur um den Gamestate, der durch den Host verwaltet wird, synchron zu halten.

Der prinzipielle Aufbau des Systems kann durch das Verteilungsdiagramm Abbildung 4.2 symbolisiert werden. In dem Verteilungsdiagramm ist weiterhin

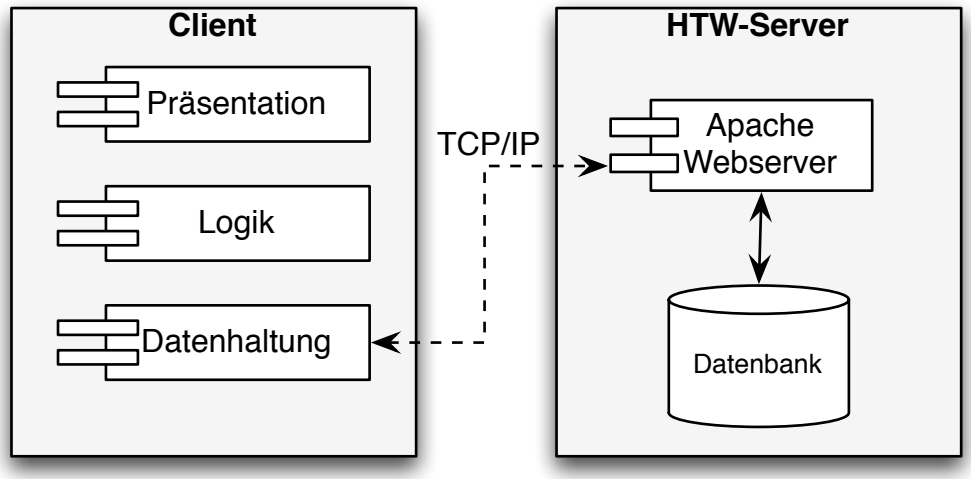

Abbildung 4.2.: Verteilungsdiagramm

ersichtlich, dass die Anwendung (Client) in 3 Schichten, entsprechend dem MVC-Architekturmuster eingeteilt ist. Diese Form der Architektur empfiehlt auch Apple, da viele Objekte innerhalb eines Programmes eine Wiederverwendbarkeit aufweisen und auch einfacher zu skalieren sind. Die 3 Schichten setzen sich aus

- Präsentationsschicht - darstellende Schicht
- Anwendungsschicht - kontrollierende Schicht
- Datenhaltungsschicht - Daten haltende Schicht

zusammen. Jedoch ist das von Apple empfohlene Architekturmuster leicht verändert worden in Bezug auf die Kommunikation zwischen den Schichten.

Beim traditionellen Design (Vgl. Abbildung 4.3) wird aus einer Benutzeraktion in der Viewkomponente ein Event generiert und an den Controller geschickt. Dieser aktualisiert dann entweder die Viewkomponente oder das Datenmodel. Das Datenmodel benachrichtigt dann alle Observer[56], die an diesem Objekt registriert sind. Falls der Observer eine Viewkomponente ist, wird diese auch direkt aus dem Model-Objekt aktualisiert.

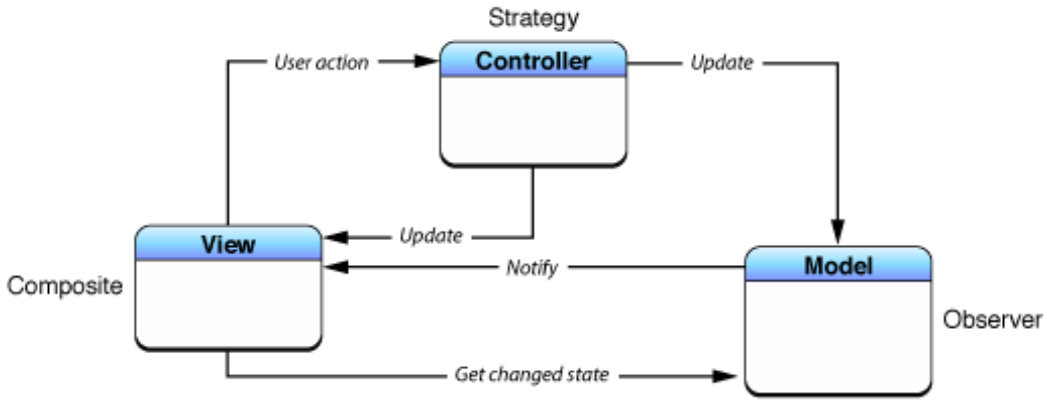

Abbildung 4.3.: traditionelles MVC-Pattern; Quelle: [7]

Im Gegensatz dazu, wird beim Cocoa-Ansatz (Vgl. Abbildung 4.4) erst wieder der Controller informiert, der dann die Änderung an die Viewkomponenten weiterreicht. In diesem Ansatz erfolgt die Kommunikation zwischen den Schichten nur über den Controller, weil so die Wiederverwendbarkeit der Objekte in den Schichten erhöht wird. Trotzdem ist durch Einsatz der Bindings-Technologie es auch hier möglich aus dem Model-Objekt die Viewkomponente direkt anzusprechen.

Als Schlussfolgerung aus der MVC-Architektur werden in den nächsten Abschnitten die einzelnen Schichten genauer erklärt.

[56] zu dt.: Beobachter

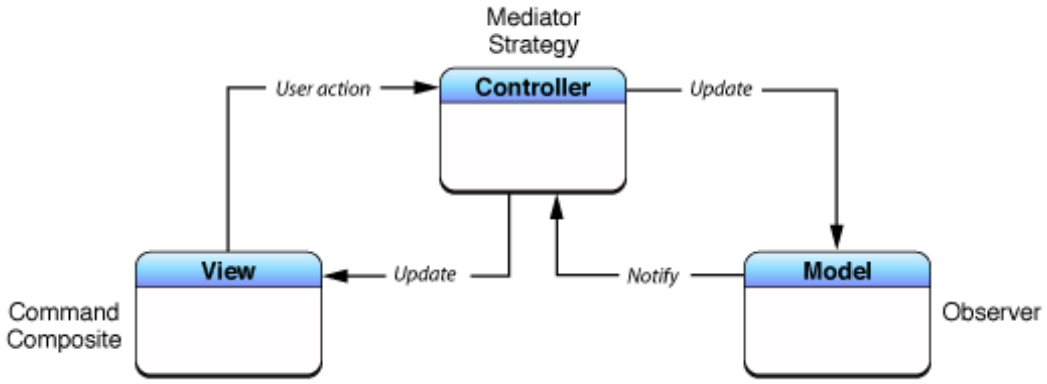

Abbildung 4.4.: Cocoa MVC-Pattern; Quelle: [7]

4.2. Datenmodell

In der Datenhaltungsschicht werden die Model-Objekte definiert. Sie kapseln die Daten, die für die Anwendung gebraucht werden und definieren außerdem die Zugriffe auf die Daten. Um sich nach den Cocoa MVC Design Pattern zu richten, werden die Model-Objekte explizit ohne Verbindung zur grafischen Benutzeroberfläche entwickelt. Genauer gesagt sollen Model-Objekte höchstens von anderen Model-Objekte anhängig sein. Durch diesen Ansatz gelingt es eine hohe Wiederverwendbarkeit der Model-Objekte zu erreichen[7].

Aus den Anforderungen an das System ist zu entnehmen, dass es 2 Arten der Datenhaltung geben soll. Durch den Anspruch an eine Online-Highscoreliste, lässt sich schlussfolgern, dass eine persistente Datenhaltung im Internet stattfinden muss. Dagegen befinden sich die restlichen Daten, die zur Umsetzung benötigt werden lokal auf den Geräten. Abbildung 4.5 skizziert dies.

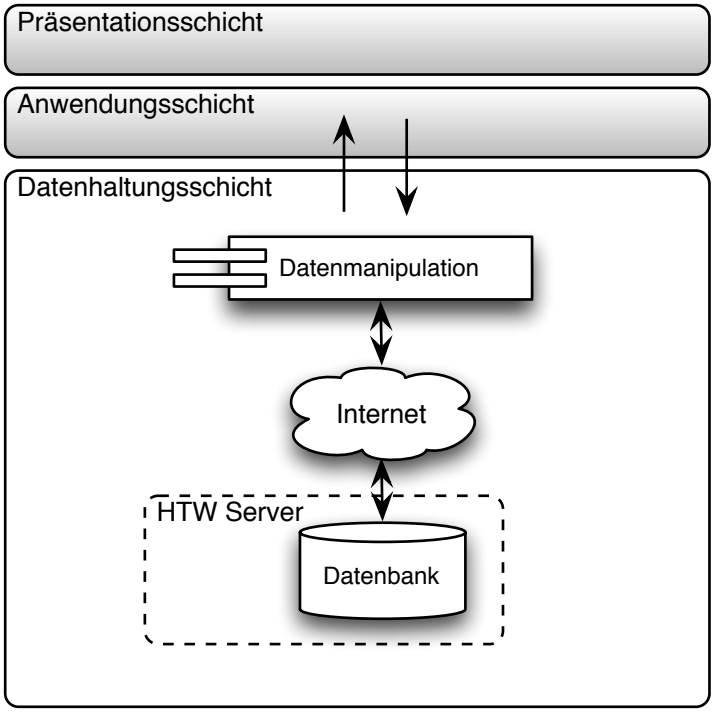

Abbildung 4.5.: Datenhaltungsschicht

4.2.1. entfernte Datenhaltung

Um die Speicherung von persistenten Daten zu gewährleisten, haben sich die Datenbankmanagementsysteme gegenüber anderen Methoden, wie Speicherung als XML-Datei als solide erwiesen. Durch Datenbanken ist es möglich große Mengen an Daten zu speichern, und diese auch untereinander zu verknüpfen bzw. in Beziehung zu setzten. Die am häufigsten vertretende Art der Datenbanken, bilden die relationalen Datenbanken. Bei relationalen Datenbanken werden Daten in Tabellenform (Relationen) dargestellt, wobei die Daten einer Spalte als Attribute bezeichnet werden. Den Zusammenschluss verschiedener Attribute einer Spalte nennt man wiederum Tupel. Das Abfragen oder Manipulieren von Daten geschieht mit Hilfe der SQL[57] Sprache. Die theoretische Grundlage dieser Sprache bildet die relationale Algebra, die durch mengenorientierte und tupelorientierte Operationen komplexe Informationen aus

[57]Structured Query Language

Relationen ableiten kann. Um Redundanzen und Anomalien, sowie die Konsistenz der Daten zu sichern, müssen die Relationen normalisiert[58] sein.

Um die persistente Datenhaltung für den Highscore mittels einer relationalen Datenbank zu erreichen, genügt lediglich eine Relation (Vgl. Tabelle 4.1).

Tabelle 4.1.: Struktur der Relation highscore

Feld	Typ	Null	Standard
id	int(11)	Ja	NULL
name	varchar(20)	Ja	
points	int(11)	Ja	
longitude	varchar(20)	Ja	0
latitude	varchar(20)	Ja	0

In dieser Relation werden lediglich der Name, die erreichte Punktzahl sowie die Positionsdaten, gegeben durch die Longitude und Latitude gespeichert. Anhand des Attributs *points* kann eine Liste mit absteigender Punktzahl generiert werden. Das Attribut *id* bildet dabei den Primärschlüssel, der aber keine weitere Rolle spielt. Durch das Vorhandensein nur einer Relation und der Unterscheidung anhand des Namens, ist jedoch keine echte Spieleridentifizierung möglich. Um eine echte Identifizierung zu gewährleisten, wäre wieder die Implementierung eines Spielservers mit einhergehender Anmeldung des Spielers vonnöten.

4.2.1.1. Der Datenzugriff

Der Zugriff auf die Daten erfolgt mit Hilfe einer Klasse, die alle relevanten Operationen auf die Datenbank vornimmt. Die Klasse **Database** bietet Methoden, die zum Abfragen oder Einfügen von Daten in die Datenbank vorgesehen sind. Um aus der Anwendungsschicht mit dem Highscore arbeiten zu können, müssen folgende Methoden implementiert werden:

fetchDataFromDatabase: diese Methode hat die Aufgabe, alle Datensätze der Relation auszulesen

[58] Edgar F. Codd definierte 4 Normalformen

getPlacement: anhand der Punktzahl des Spieler, die Platzierung in der Highscoreliste ausgeben

insertData: points: zum Eintragen eines Highscores mit Name und Punktzahl

insertData: points: withLocation: zum Eintragen eines Highscores mit Name, Punktzahl und Positionsangabe

So können Klassen aus der Anwendungsschicht auf die Daten der Datenbank zugreifen indem sie die bereitgestellten Methoden der *Database* Klasse verwenden. Dadurch wird die Komplexität der Zugriffe gegenüber der Klassen der Anwendungsschicht verborgen. Eine Validierung der Daten muss nicht durchgeführt werden, da der Spieler nur die Möglichkeit hat seinen Namen einzutragen. Es muss nur sichergestellt werden, dass bei der Positionsbestimmung korrekte Daten ermittelt wurden.

4.2.2. lokale Datenhaltung

Die lokale persistente Datenhaltung findet viel Anwendung, um beim Start eines Spiels an der letzten Position weiter spielen zu können. Aus dem Spielkonzept heraus besteht prinzipiell hier keine Anforderung darin Daten auf dem Mobiltelefon persistent zu speichern. Eine dauerhafte Speicherung würde dann Sinn machen wenn ein Spiel mehrere Minuten dauert, was aber nicht der Fall ist. In der Regel dauert ein Spiel wenige Sekunden bis zu ein paar Minuten. Falls doch mit dem Spiel aufgehört werden muss, kann man die Pause Funktionalität verwenden.

Klassen

Wie bereits erwähnt werden Daten und Verhalten bei Objektorientierte Programmiersprachen in Objekten gekapselt. Für den Spielverlauf werden Model-Objekte benötigt, die den Spieler repräsentieren und die Spielobjekte, die es gilt einzusammeln bzw. auszuweichen. Da beide Objekte im Grunde über gleiche Eigenschaften verfügen, leiten sie sich aus der Klasse *GameObject* ab (Vgl.

Abbildung 4.6). Die Klasse **GameObject** definiert Attribute, wie die Position (*x_pos, y_pos, z_pos*), Anzahl der Punkte (*points*), wobei diese Variable eine unterschiedliche Aufgabe in den Klassen **Enemy** und **Player** erfüllt. Außerdem kann die Geschwindigkeit (*velocity*), die Größe (*size*) und die Textur (*texture*) definiert werden.

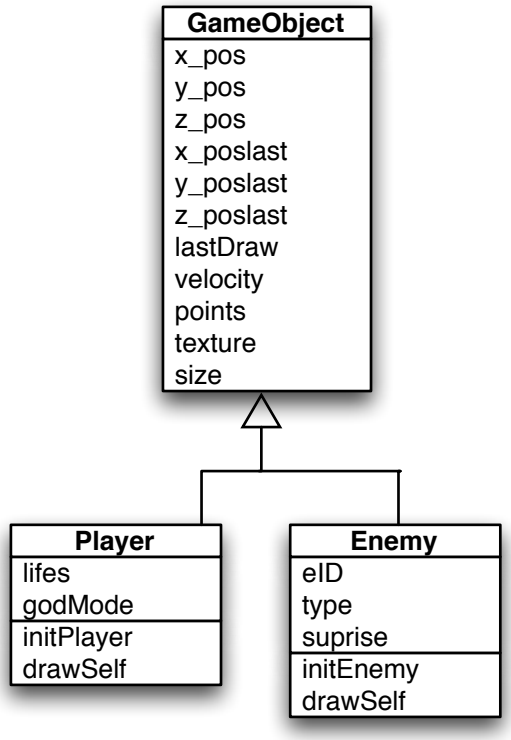

Abbildung 4.6.: Model-Objekte für Spieler und Gegner

Eine Unterscheidung zwischen **Player** und **Enemy** geschieht durch zusätzliche Attribute.

die Player Klasse: Um den Spieler vollständig beschreiben zu können, sind zusätzlich die Anzahl seiner Leben (*lifes*), sowie sein Verwundbarkeitszustand (*godMode*) definiert. Zu den konventionellen Setter und Getter Methoden, gibt es eine Methode, die den Spieler mit Ausgangswerten initialisiert (*initPlayer*) und eine weitere die beschreibt, wie der Spieler zu zeichnen (*drawSelf*) ist.

die Enemy Klasse: Für die *Enemy*-Klasse werden noch zusätzliche Instanzvariablen gebraucht, da die Objekte auf dem Spielfeld sehr unterschiedliche Eigenschaften haben. Durch *type* und *surprise* werden die Art und der Bonus

des Objektes auf dem Spielfeld beschrieben. Auch hier gibt es Methoden zur Initialisierung und Darstellung.

Um Spieler, Gegenspieler und Spielobjekte zu erzeugen, werden Instanzen der Klasse **Player** gebildet, sowie Instanzen der Klasse **Enemy**.

Audiodateien

Um ein gewisses Feedback auf den Nutzer zu erzeugen, werden Kollisionen mit Spielobjekten durch eine Soundausgabe unterstrichen. Diese Dateien müssen auf dem Gerät vorhanden sein und folgende Eigenschaften erfüllen:

- vom Typ .caf, .aif oder .wav
- Format: PCM oder IMA/ADPCM (IMA4)
- die Dauer muss kürzer sein als 30 Sekunden

Als Folge ist die Implementierung einer Klasse **SoundEffect** notwendig, die das Abspielen und Laden der Audiodateien kapselt (Vgl. Abbildung 4.7).

Abbildung 4.7.: Klasse zur Handhabung der Audiodateien

Texturen

Durch den Anspruch einer realistischen 3D-Welt müssen Bilder für die Texturierung der Objekte persistent vorhanden sein. Diese können in gewöhnlichen Formaten wie Bitmap oder JPEG[59] vorliegen. Durch die begrenzten Ressourcen des iPhones sollten komprimierte Bildformate verwendet werden. Da der PowerVR MBX Grafikchip des iPhones PVRTC komprimierte Bilder unterstützt, empfiehlt auch Apple die Verwendung dieser Kompressionsmethode. Voraussetzungen um ein Bild ins PVRTC-Format zu bringen sind:

[59] Joint Photographic Experts Group

1. Höhe und Breite müssen identisch sein
2. Höhe und Breite müssen einer Potenz von 2 entsprechen
3. Die Quelldatei muss entweder als PNG[60], JPEG oder TIFF[61] vorliegen

Im Durchschnitt nimmt ein PVRTC komprimiertes Bild nur 15% des Speicherbedarfs eines Bitmaps ein.

Strukturen

Des weiteren werden für die Darstellung von Objekten durch OpenGL, beschreibende Daten zu den Objekten benötigt. Diese Beschreibung der Objekte (Würfel, Kegel, Kugel) findet in Header-Dateien durch Arrays statt (Vgl. Abbildung 4.8). Diese Header werden dann von den Klassen **Player** und **Enemy** eingebunden. Auch die Beschreibung des Spielfelds befindet sich in einer Header-Datei.

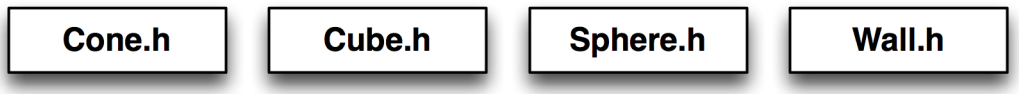

Abbildung 4.8.: Header-Dateien für Objektformen

Aufgrund der Repräsentation der Objekte innerhalb von OpenGL durch Polygone, ist es sinnvoll elementare Strukturen zu schaffen. Daher werden Strukturen erzeugt um Polygone, Vertexe, Faces und Normale besser beschreiben zu können (Vgl. Abbildung 4.9). Diese Strukturen werden dann von den Header-Dateien der Spielobjekte benutzt um die Beschreibung der Objekte besser vollführen zu können.

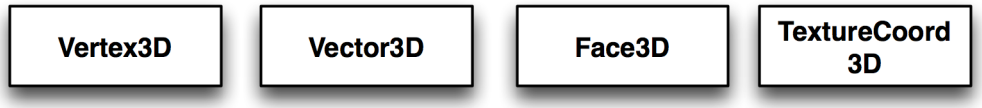

Abbildung 4.9.: Strukturen zum Beschreiben der OpenGL Objekten

[60] Portable Network Graphics
[61] Tagged Image File Format

4.3. Anwendungslogik

Die Anwendungsschicht positioniert sich zwischen der Präsentations- und Datenschicht. Ihre Aufgabe ist es den Vermittler zwischen Präsentation und Datenhaltung zu spielen. Sie stellt also sicher das View-Objekte Zugriff auf das Datenmodell haben. Die Objekte in der Anwendungslogik neigen eher dazu nicht wiederverwendbar zu sein, da sie die Applikation initialisieren und für den Lebenszyklus der Anwendung die Verantwortung tragen. Außerdem hält sie die ganze Logik, die zum Ablauf eines Programmes nötig ist. Die Daten, die diese Schicht produziert, werden entweder dargestellt oder sie werden in Model-Objekte gespeichert.

Den Kern der Anwendungsschicht bilden die Gewinnung und Auswertung von Sensordaten, sowie die Logik, die zur Ausführung des Spiels, notwendig ist. Da bis auf den Austausch des Gamestate kein Unterschied in der 1-Spieler Variante zur 2-Spieler Variante besteht, wird im folgenden immer auf die 2-Spieler Variante Bezug genommen. Somit ist der Informationsaustausch zwischen den Geräten ein weiterer Kernpunkt, den die Anwendungsschicht übernimmt.

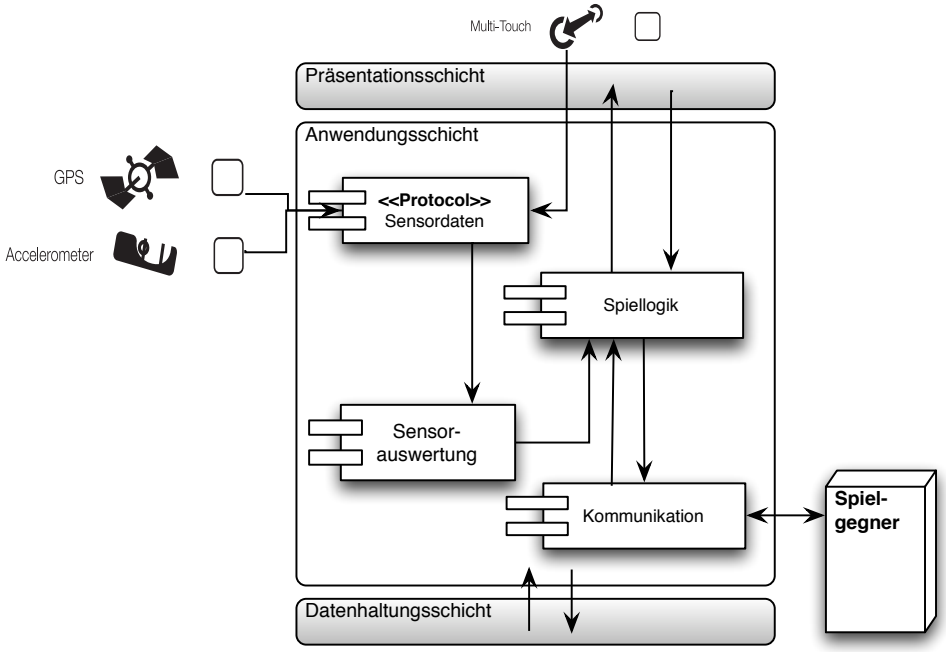

Abbildung 4.10.: Abläufe der Anwendungsschicht

Abbildung 4.10 zeigt die schematisch dargestellte Funktionsweise. Aus der Präsentation kommen Eingabedaten als Folge von Bildschirmberührungen, während zusätzlich die Sensordaten der anderen Sensoren ermittelt werden. Anhand dieser Daten findet eine Auswertung statt und deren Ergebnisse werden an die Spiellogik übermittelt. Die Spiellogik ist zentraler Bestandteil der Anwendungsschicht, da sie auch die Kommunikation zum Spielpartner regelt und weiterhin, die Präsentationsschicht mit neuen Informationen versorgt auf Basis der Model-Objekte.

4.3.1. Der Gameloop

Der Gameloop ist wichtigstes Element im Spielentwurf wenn es um den Ablauf des Spiels geht, da er die Steuerung hinter dem Spielgeschehen bildet. Der Gameloop wird durch die Klasse **GameController** implementiert und hat folgende Aufgaben:

- Auswertung der Sensordaten
- Gamestate senden und empfangen
- Status bzw. Position der Objekte aktualisieren
- Status und Position der Spieler aktualisieren
- Kollisionen von Spieler und Spielobjekte überprüfen
- Inhalte auf dem Display darstellen

Aus Abbildung 4.11 kann der prinzipielle Ablauf des Spielgeschehens entnommen werden.

Bevor der Gameloop mit seiner Arbeit anfangen kann, wird jedoch alles was innerhalb des Spiels gebraucht wird initialisiert. Neben der grafischen Oberfläche werden außerdem die Sounddateien in den Speicher geladen. Weiterhin werden die Spieler sowie die Spielobjekte initialisiert. Um den Datenaustausch zu gewährleisten werden auch die dazu benötigten Klassen instantiiert.

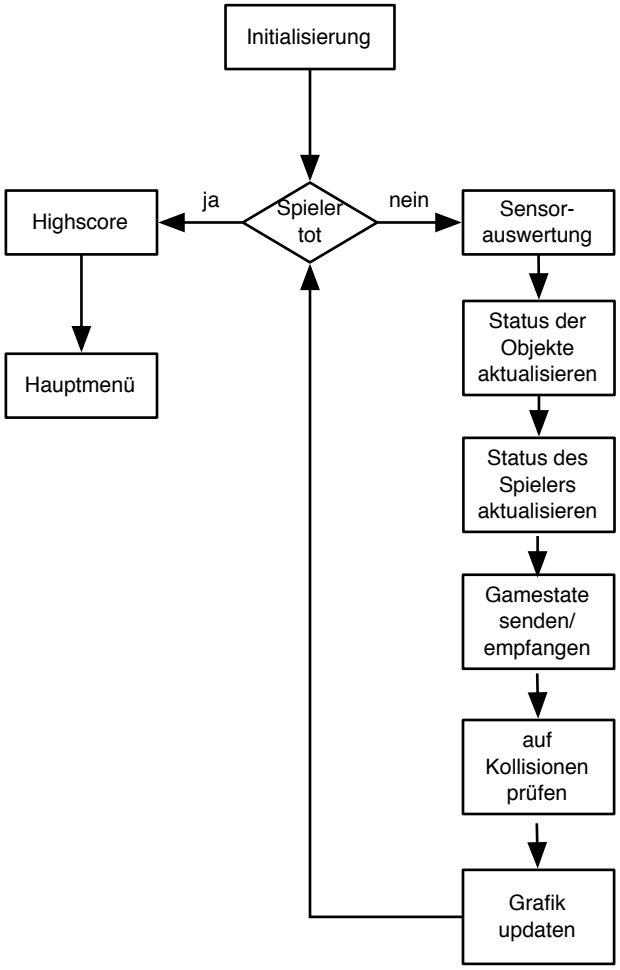

Abbildung 4.11.: Der Gameloop

Zur Initialisierung der Spielobjekte werden Pseudozufallszahlen[62] genutzt, um den Typ des Spielobjektes, die Startposition und die Geschwindigkeit festzulegen. Für den Fall als Typ eine *Überraschungsbox* zu erhalten, wird auch hier mittels Zufallszahlengenerator die Art der Überraschung erzeugt. Hier kommt es darauf an, einen geeigneten Generator zu verwenden um den Eindruck von echt zufälligen Spielobjekten mit verschiedenen Geschwindigkeiten zu bekommen. Das zufällige Erzeugen der Startposition hat den Hintergrund, den Eindruck zu vermeiden, dass die Spielobjekte in Intervallen ankommen. Abbildung 4.12 spiegelt den Aufbau des Spielfeldes aus der Draufsicht wieder. Zu Erkennen ist, dass das Spielfeld eine Länge von 70 Einheiten und eine Breite von 4 Einheiten hat. Die Y-Achse ist zu vernachlässigen da, das Spielgeschehen nur

[62] Zufälligkeit, die mathematisch vorhersagbar ist

auf einer Ebene stattfinden wird. Die Startposition des Spielers liegt bei 0,-2 bei einer Bewegungsfreiheit nur auf der X-Achse. Im oberen Bereich kann man das Gebiet der Startpositionen (-30 bis -55) für die Spielobjekte begutachten. Diese Objekte bewegen sich nach der Initialisierung entlang der Z-Achse auf den Spieler zu. Die Kameraposition befindet sich hinter dem Spiel in einer erhöhten Position (Y-Achse).

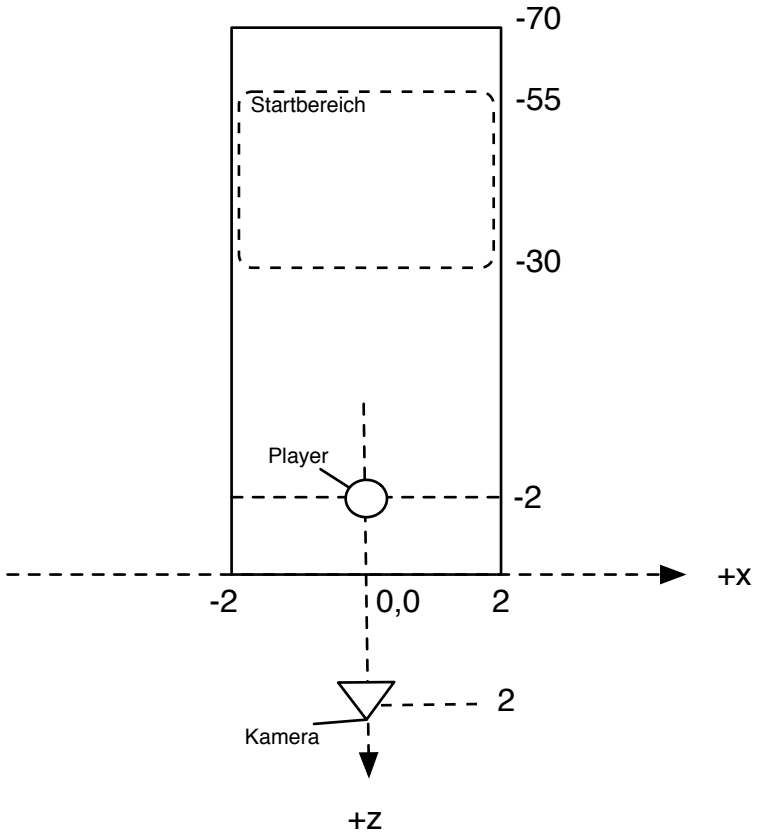

Abbildung 4.12.: Aufbau des Spielfelds

Jeder Start des Gameloops findet mit der Überprüfung ob die Spieler am Leben sind statt. Sollten die Spieler tot sein, wird der Gameloop verlassen und ein Objekt der Klasse **AddHighscore** instantiiert, welches die Platzierung im Highscore anzeigt und falls gewollt den Eintrag in die Datenbank vornimmt. Danach ist das Spiel zu Ende. Abbildung 4.13 zeigt ein Sequenzdiagramm mit dem Ablauf für einen Datenbankeintrag.

Für den Fall, dass die Spieler noch am Leben sind, werden zuerst die Sensordaten ermittelt (Vgl. Kapitel 4.3.2 S. 69). Die Positionsänderung des Spielers

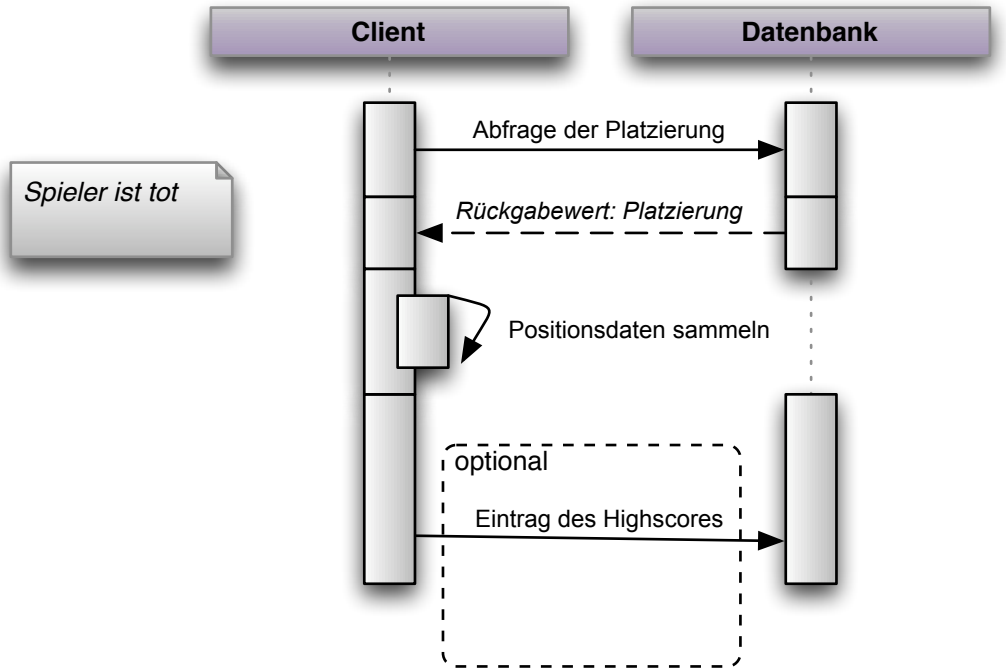

Abbildung 4.13.: Sequenzdiagramm für Highscoreeintrag

geschieht anhand der Werte, die aus dieser Auswertung erfolgt. Da als Ergebnisse, Werte zwischen -1 und +1 zu erwarten sind, können diese als Wert für die Geschwindigkeit des Spielers in Form eines Vektors $\vec{v}$ interpretiert werden. Die neue Position des Spielers ergibt sich dann aus folgender Gleichung:

$$Pos_{neu} = Pos_{alt} + \vec{v}$$

Ähnlich geht die Positionierung der Spielobjekte von statten. Neben der zugeteilten Geschwindigkeit des Objektes fließt zudem noch ein *SpeedUp*, der durch eine Überraschungsbox entstanden sein kann, sowie ein Wert der abhängig vom gespielten Level ist, mit ein.

$$Pos_{neu} = Pos_{alt} + speedUp + level * 2 + \vec{v}$$

Durch die neu errechneten Daten hat sich der Gamestate (Position der Spieler und Spielobjekte) verändert. Dieser neue Gamestate muss anschließend mit dem Gegenspieler synchronisiert werden.

Da sich die Objekte jetzt an neuen Orten befinden, wird eine Kollisionsabfrage zwischen den Spielern und den Spielobjekten ausgeführt. Kollisionen haben wieder eine Auswirkung auf den Gamestate, welcher dann wieder synchronisiert werden muss. Nach der Abhandlung der Logik im Spiel erfolgt die grafische Ausgabe des Gamestate. Diese Schleife wird so oft durchlaufen, bis ein Anwender das Spiel beendet hat oder beide Spieler 0 Leben haben.

4.3.2. Extraktion der Sensordaten

Die Gewinnung und Auswertung der Sensordaten stellt einen wichtigen funktionalen Aspekt in der Anwendungsschicht dar. Nur allein durch die generierten Daten des Beschleunigungssensors ist eine Steuerung der Spielfigur überhaupt möglich. Um überhaupt durch Bedienelemente, wie Buttons, durch das Spiel navigieren zu können müssen Daten des Multitouch-Sensors ausgelesen werden. Der Positionssensorik kommt in dem Spiel die geringste Aufmerksamkeit zugute, da die Positionsdaten lediglich eine Verwendung in der Highscoreliste finden.

Zentrales Element der Sensorauswertung ist der **GameController**, der von einem **UIViewController** abgeleitet ist. Die Gewinnung der Sensordaten geschieht durch Protokolle (Interfaces), die der **GameController** implementieren muss. Um eine Zustandsänderung im Beschleunigungssensor zu registrieren muss eine Methode des Protokolls **UIAccelerometerDelegate** überschrieben werden. Diese Methode wird dann in einem vorher festgelegten Zeitintervall aufgerufen und liefert dann die momentanen Beschleunigungswerte des Gerätes.

Analog verhält es sich mit der Erfassung der Positionskoordinaten. Eine Instanz des ViewControllers **AddHighscore** wird im GameController erzeugt. Die AddHighscore Klasse bietet die Möglichkeit sich in die Highscoreliste einzutragen. Hier wäre es auch möglich das Protokoll für die Positionsbestimmung einzubinden, da aber der Prozess der Informationsgewinnung, sowie die Initialisierung komplexer ist, wird die Logik in eine eigene Klasse (LocationController) gekapselt. Der **LocationController** implementiert das Protokoll **LocationControllerDelegate**. Hier muss ebenfalls eine Methode des Protokolls überschrieben werden um die Positionsdaten zu bestimmen.

Zur Auswertung der Berührungen auf dem Display spielt die **UIView** eine wichtige Rolle. In dieser Klasse findet nicht nur die grafische Ausgabe statt, sondern sie nimmt auch die Events, die aus Berührungen mit dem Display resultieren entgegen. Die Verarbeitung dieser Daten findet im verwaltenden Controller der UIView, dem GameController statt. Abbildung 4.14 zeigt den Zusammenhang der Klassen, die zur Auswertung von Sensordaten eine Rolle spielen.

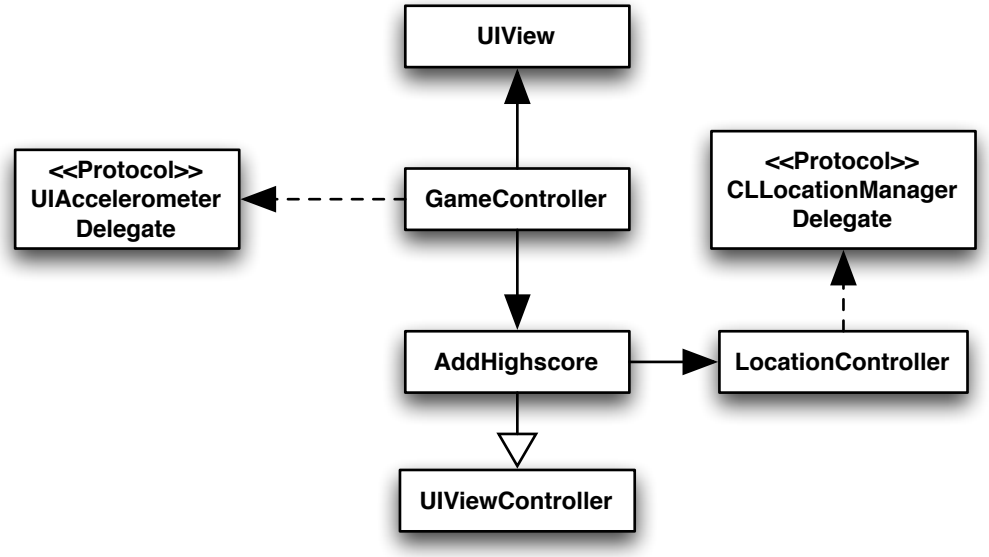

Abbildung 4.14.: Zusammenhang der Klassen zur Auswertung von Sensordaten

4.3.3. Kommunikation zwischen Client und Host

Aus der Analyse zum Systementwurf ging hervor, dass die Kommunikation über UDP-Sockets stattfinden muss. Da diese Art der Kommunikation verbindungslos ist, braucht der Host sich nur an einen Port binden und dann auf ankommende Pakete warten. Dieser antwortet dann mit Hilfe der Ursprungsadresse (Adresse vom Client) aus dem Paketheader. Abbildung 4.15 soll dies verdeutlichen. Der Client dagegen hat anfänglich keine Ahnung wohin er seine Pakete schicken muss.

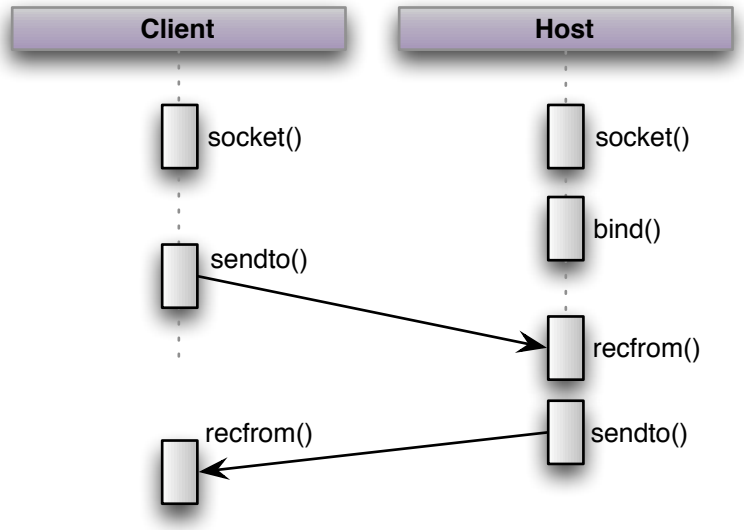

Abbildung 4.15.: Sequenzdiagramm einer UDP-Socket Verbindung

Um eine Verbindung zwischen Client und Host herzustellen, bedarf es des Entwurfs einiger Klassen. Der **Balls_And_CubesViewController** repräsentiert das Hauptmenü des Spiels. Hier soll der Nutzer die Auswahl haben ein Spiel zu erstellen (Host) oder dem Spiel beizutreten (Client). Diese Möglichkeiten werden durch Instanzen des **HostController** und des **ClientController** im Balls_And_CubesViewController implementiert. Der HostController hat die Aufgabe einen Socket zu erstellen und mit Hilfe von Bonjour (Vgl. 2.3.7 S. 26) durch Instanziierung der Klasse **NSNetService** zu veröffentlichen. Ähnlich verhält es sich mit dem ClientController, der eine Klasse vom Typ **NSNetServiceBrowser** instantiiert. Dieses Objekt soll eine Auflistung der verfügbaren Spiele liefern. Jedes dieser verfügbaren Spiele ist ein Objekt vom NSNetService und enthält die Adresse des Hosts. Durch die Implementierung entsprechender Protokolle, **HostControllerDelegate** und **ClientControllerDelegate**, kann der Balls_And_CubesViewController darüber informiert werden, das ein Spielpartner gefunden wurde. Dessen Aufgabe ist es nun auf Clientseite ebenfalls ein Socket zu erstellen und mit dem Host Verbindung aufzunehmen. Weiterhin wird ein Objekt vom **GameController** erzeugt und die entsprechenden Methoden zur Initialisierung als Host oder Client vorgenommen. Abbildung 4.16 zeigt den Zusammenhang der Klassen und Protokolle zur Spielpartnerfindung.

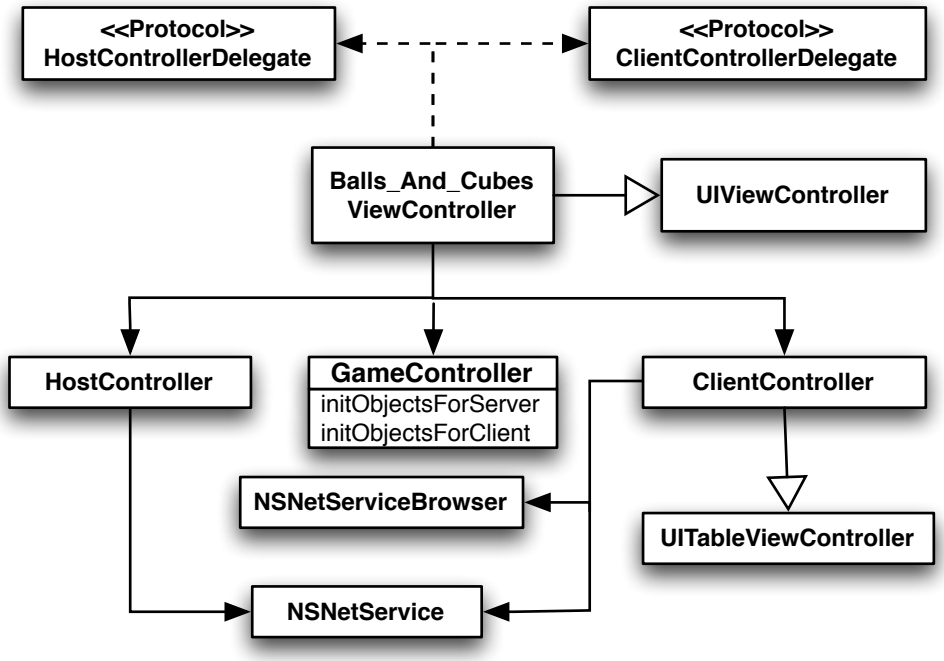

Abbildung 4.16.: Zusammenhang der Klassen zum Start des 2-Spieler Spiels

Durch die Unterstützung von Bonjour entfällt das gegenseitige Finden der Spielpartner durch einen UDP Broadcast des Hosts.

Weiterhin ist es sinnvoll die Logik und Komplexität der Erstellung von Socketverbindungen, sowie das Senden/Empfangen von UDP-Paketen in Klassen zu kapseln. Um doch ein gewisses Maß an Vertrauen in die UDP-Verbindung setzten zu können werden Konzepte von TCP in den Aufbau der Klassen übernommen.

Die Klasse **Connection** verwaltet die Erstellung, die Aufrechterhaltung und das Beenden der Verbindung zwischen den Spielpartnern. Obwohl der UDP-Host keine virtuelle Verbindung zum Client hält, kann doch eine erreicht werden, da es sich nur um einen Client handelt. Der Host, als auch der Client kennen die Adresse ihrer Spielgegner durch eine Instanz der Klasse **SocketAddress**. Diese Adresse kann auf Hostseite nach dem Empfang des ersten Paketes vom Client gesetzt werden. Da aber der UDP-Host dazu neigt von überall Pakete zu empfangen, erweist es sich als nützlich eine **protocolId** zu implementieren. Durch Implementierung dieser, nimmt der Host nur Pakete mit entsprechender ID an. Weiterhin muss erkannt werden, wenn der Spiel-

gegner das Spiel verlässt. Auf Clientseite geschieht das mit Hilfe eines Protokolls, das erkennt wenn der vom Host veröffentlichte Dienst beendet wurde. Auf der Hostseite muss dieses über ein Timeout geregelt werden. Wenn die Instanzvariable **timeout** einen bestimmten Schwellenwert überschreitet, kann davon ausgegangen werden, dass der Client das Spiel verlassen hat. Durch die

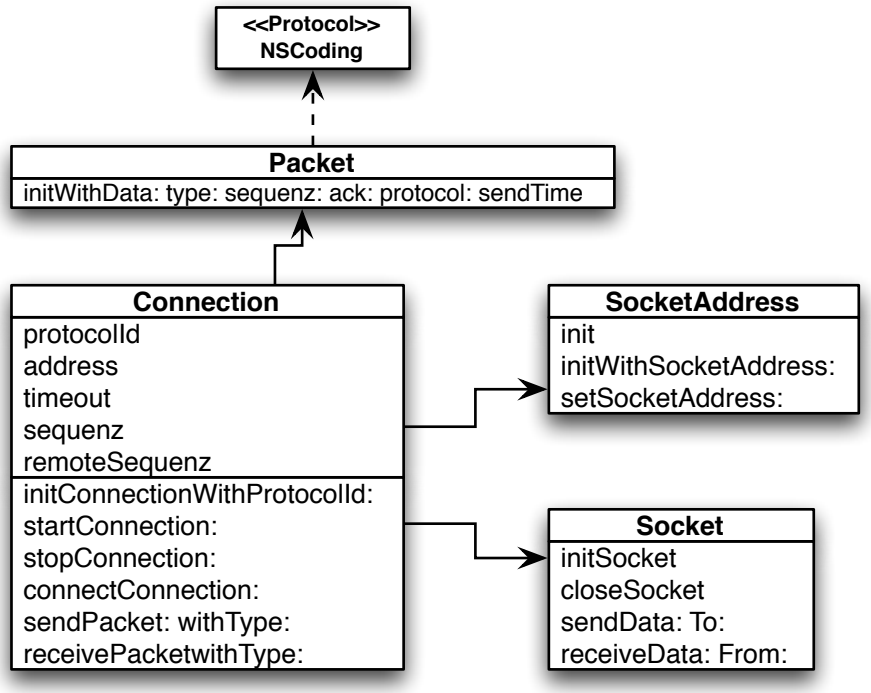

Abbildung 4.17.: Klassendiagramm zur Kommunikation

Natur von UDP, kann es vorkommen das Pakete in unterschiedlicher Reihenfolge eintreffen. Um dies zu vermeiden wird eine Instanzvariable **sequenz** in die Klasse Connection eingefügt. Jedes Paket bekommt vor dem Senden eine Sequenznummer, die um eins höher als beim letzten Paket ist, mit. Um zu wissen welche Sequenznummer der Spielgegner hat, ist zudem noch eine **remoteSequenz** zu implementieren, die beim Empfang eines Paketes durch die Sequenznummer gesetzt wird. Ein Paket wird durch eine Instanz der Klasse **Packet** gebildet. In dieser Klasse werden der Typ, die Sequenznummer, die ProtokollID und die eigentlichen Nutzdaten (Payload) durch Instanzvariablen festgelegt. Diese Klasse muss vor dem Senden serialisiert werden, durch Implementierung der Methoden des **NSCoding**-Protokolls. Die Erstellung eines Sockets, sowie das eigentliche Senden und Empfangen geschieht in einem

Objekt der Klasse **Socket**, welches vom Connection-Objekt erzeugt wird. Abbildung 4.17 verdeutlicht diese Zusammenhänge.

Eine weitere Überlegung ist es in welchem Takt das Senden der Pakete erfolgen soll. Hier gibt es die Möglichkeit, es entweder Event-gesteuert oder nach einem konstanten Zeitintervall zu gestalten. Aufgrund der Realtime-Anforderung erweisen sich das Übermitteln des Gamestates in konstanten Zeitintervallen (state-based) als die bessere Methode. Als Zeitpunkt für das Senden, kann ein Logiktick (ein Durchlauf des Gameloop) genommen werden. Erwartungsgemäß findet das Senden von Paketen ca. 30 mal in der Sekunde statt. Durch fehlende Flusskontrolle, kann es vorkommen, dass Pakete sich beim Empfänger in einer Warteschlange sammeln, da diese nicht schnell genug abgearbeitet werden können. Nach dem FIFO[63] Prinzip werden die Pakete dann abgearbeitet. Da es aber beim Empfang nur auf das aktuellste Paket (feststellbar anhand der Sequenznummer) ankommt, müssen die Pakete solange verarbeitet werden bis die Warteschlange leer ist. Abbildung 4.18 zeigt den schematischen Ablauf des Empfangs innerhalb des Gameloops.

[63] First In First Out

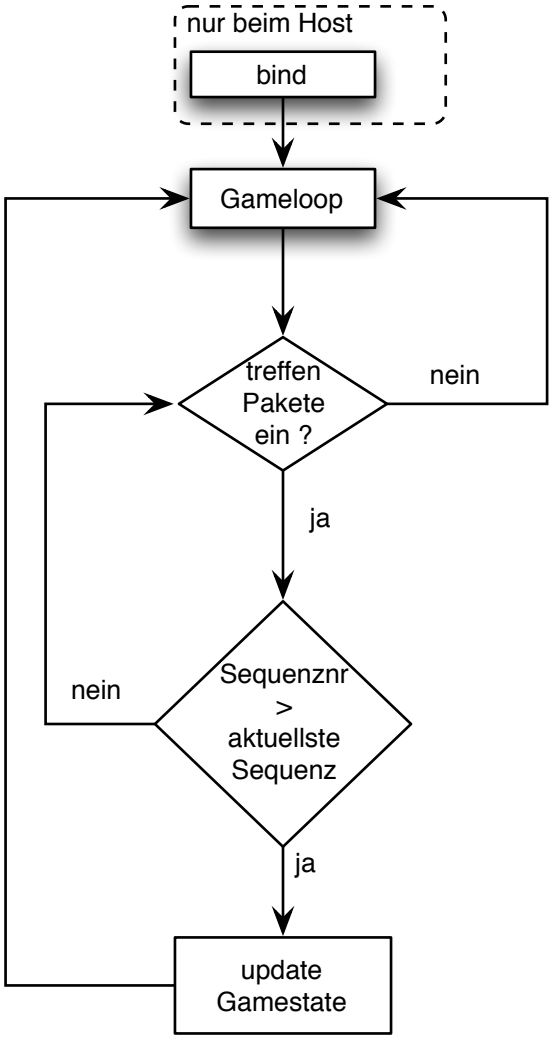

Abbildung 4.18.: Ablauf des Empfangs eines Paketes

4.4. Präsentation

Eine nicht zu unterschätzende Rolle besitzt die grafische Präsentation. Aus diesem Grund sollte man sich vorher ein paar Gedanken machen die zur Entwicklung des User-Interface beitragen. Die Repräsentation entspricht der sogenannten View Komponente im Model-View-Controller Design Pattern und ist im Grunde nur eine Schnittstelle zum Benutzer, denn sie präsentiert die Inhalte auf dem User-Interface und ist weiterhin dafür verantwortlich die Benutzereingaben an den steuernden Controller weiterzuleiten, oder auch den umgekehrten Weg dem Nutzer ein gewisses Feedback auf seine Eingaben zu geben. Abbildung 4.19 stellt dies schematisch dar.

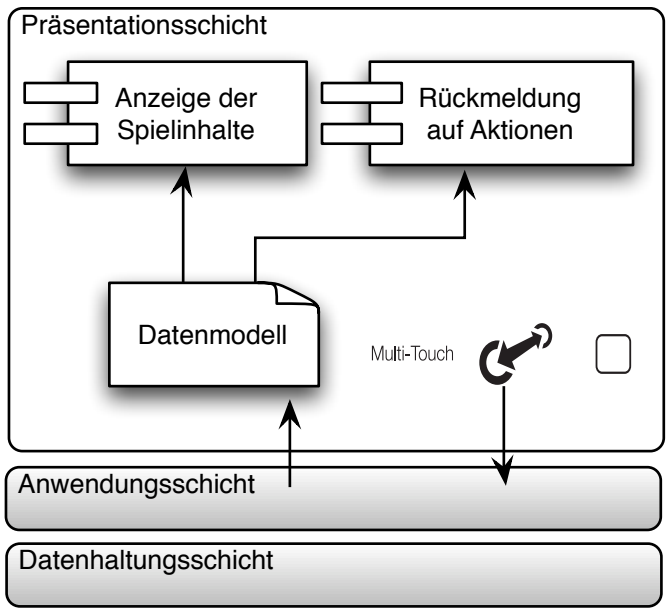

Abbildung 4.19.: Präsentationsschicht

Laut der Apple *iPhone Human Interface Guidelines* [5] wurden 3 Darstellungsweisen von Anwendungen identifiziert. Das sind zum einen die Produktivitätsanwendungen, die je nach Aufgabe die Daten meistens hierarchisch in *TableViews* darstellen, sowie *Utility* Anwendungen, die meistens ein Minimum an Nutzerinteraktion erfordern und dem Zweck der Darstellung einer Zusammenfassung von Informationen dienen. Diese Arbeit kann jedoch als *Immersive* Anwendungen ausgelegt werden. Diese Anwendungen tendieren dazu ihr eigenes User-Interface anstelle der Standard Implementierung zu verwenden, sowie

die Standard Kontrolleinheiten durch eigene zu ersetzen. Diese Variante kann sich somit viel besser ins Spiel integrieren, und wirkt nicht fremd.

4.4.1. Aufbau des User-Interface

Aus der Gegenüberstellung der Geräte ging hervor, dass das iPhone eine Darstellungsfläche von 320x480 Pixel bietet. Das hat zu Folge das unter anderem die Texteingabe gering gehalten werden soll. Vielmehr muss man den Nutzer Auswahloptionen sprich Interaktionsmöglichkeiten anbieten, um eine den Designrichtlinien entsprechende Usability zu gewährleisten. Sofern man sich eigen gestalteter Bedienelemente betätigt, muss darauf geachtet werden die Ausdehnung der Elemente ausreichend groß zu gestalten um ein daneben tippen auszuschließen. Um das Spielgeschehen ausreichend Platz zu bieten, wird die Darstellung der Spielinhalte im Breitbildformat erfolgen. Durch Abzug der Statusleiste bleiben 300 Pixel in der Höhe und 480 Pixel in der Breite zur Darstellung übrig (Vgl. Abbildung 4.20). Eine Darstellung des Spiels im Portraitmodus ist dagegen nicht zu erwägen. Hier wäre die Sicht auf das Spielgeschehen stark eingeschränkt, da nur 60% der Breite des Breitbildformats zur Verfügung stehen würden. Außerdem würde das iPhone in der Portraitansicht komisch in der Hand liegen, wenn es gilt feine Bewegungen zur Steuerung der Spielfigur auszuführen.

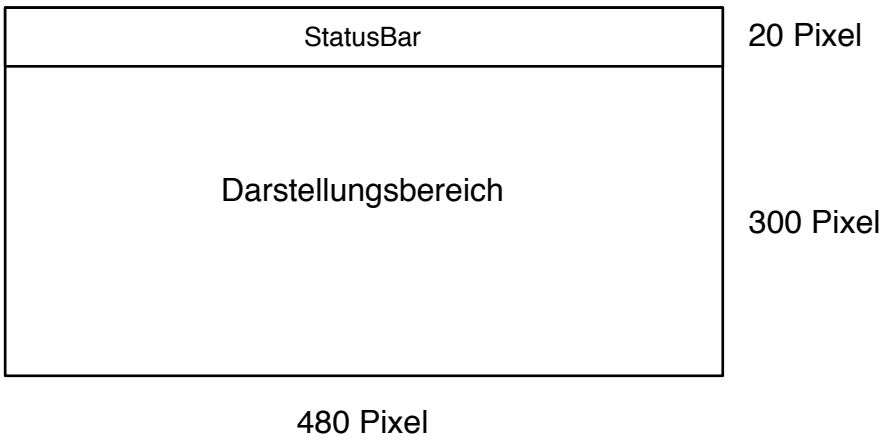

Abbildung 4.20.: Darstellungsfläche auf dem iPhone

Die Anzahl der Fenster, die eine Anwendung haben darf, ist beim iPhone auf ein Fenster beschränkt. Jedoch können auf diesem Fenster mehrere Views platziert werden. Die Views sind für folgende Aufgaben verantwortlich:

- Zeichnen und Animieren
- Layout und Subview Management
- Eventhandling

Durch Ableiten der Klasse **UIView** erzeugt man solch eine View. Für die performante Darstellung des 3D-Spielgeschehens kommt nur OpenGL|ES in Frage. Um eine View die Darstellung von OpenGL Elementen zu ermöglichen, muss der Kontext der View auf OpenGL angepasst werden. Die Klasse **glView**, die durch den **GameController** instantiiert wird, ist zur Anzeige der OpenGL-Inhalte verantwortlich. Aufgrund der Möglichkeit in einer View eine weitere View zu erzeugen, können Statusinformationen zum Spiel (Punkte, Anzahl der Leben) als eine Schicht über der OpenGL-Schicht dargestellt werden.

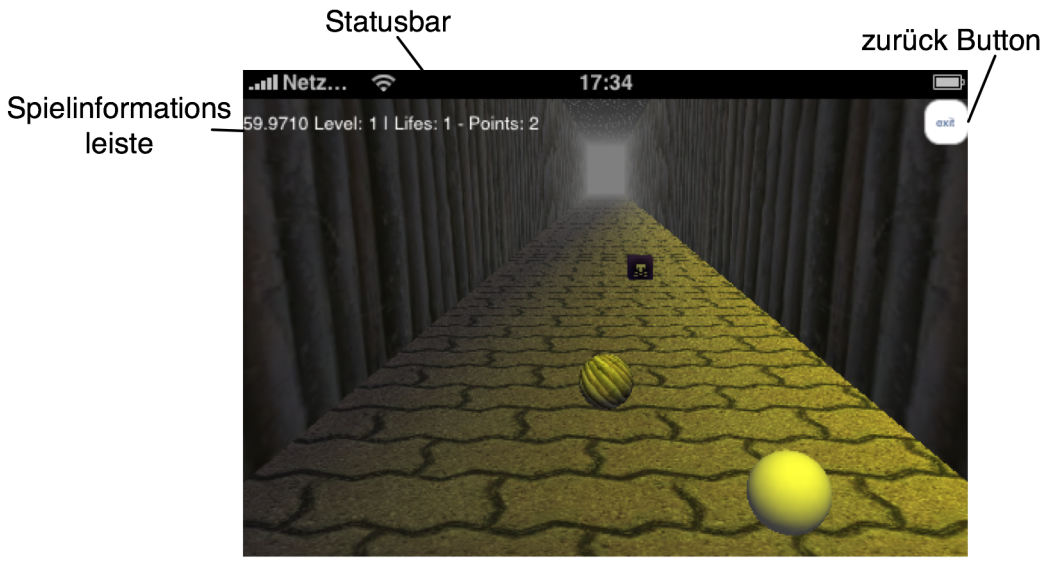

Abbildung 4.21.: Aufbau der Spieloberfläche

Ein weiterer wichtiger Aspekt in Bezug auf Views sind die ViewController der Klasse **UIViewController**, da sie eng mit den Views verbunden sind. Zu den Aufgaben von ViewControllern zählen das Erstellen und Managen von

Views, sowie die Entfernung aus dem Speicher falls wenig virtueller Speicher zur Verfügung steht. Daher ist der GameController von einem UIViewController abgeleitet. Abbildung 4.22 illustriert dies.

In dieser Arbeit werden mehrere ViewController verwendet, da jeder seine eigene Aufgabe besitzt. Grundsätzlich gibt es einen ViewController für jeden Menüpunkt plus der Spieldarstellung.

- Spieldarstellung - GameController
- Hauptmenü - Balls_And_CubesViewController
- Highscoreliste - HighscoreViewController
- Liste der Verfübaren Spiele - ClientController

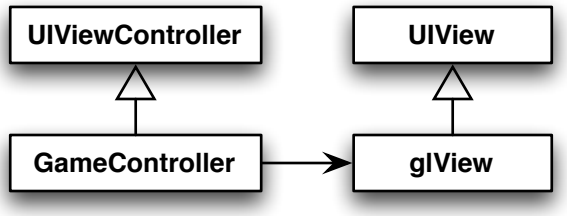

Abbildung 4.22.: Zusammenhang zwischen den GameController und der OpenGL-View

Um diese ViewController untereinander managen zu können muss eine Instanz der Klasse **UINavigationController** in der Rootklasse der Anwendung, dem **Balls_And_CubesAppDelegate** implementiert werden. Diese hat die Aufgabe die ViewController, einschließlich der instantiierten Views anzuzeigen und eine Navigation zwischen diesen zu gewährleisten. Aus Abbildung 4.23 kann der Zusammenhang zwischen den NavigationController, ViewController und dessen Views entnommen werden.

4.4.2. Bedienung

Die Bedienung erfolgt durch Berührung des Displays. Wie bereits bei der Sensorauswertung (Vgl. Kapitel 4.3.2 S. 69) erwähnt leitet die View den Event an den entsprechenden Controller weiter. Das gleiche Prinzip gilt für die Buttons oder Slider, die im Hauptmenü Anwendung finden (Vgl. Abbildung 4.24).

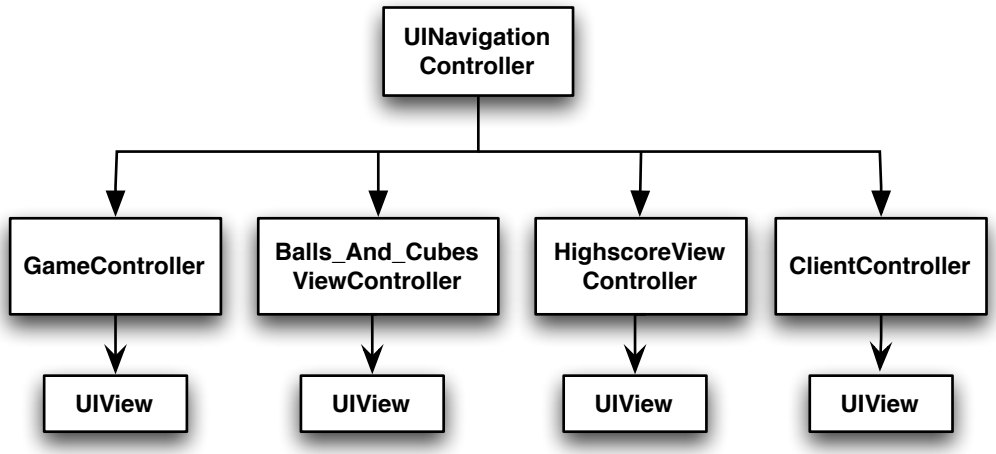

Abbildung 4.23.: Verwaltung der ViewController durch den NavigationController

Um Buttons bzw. Slider verwenden zu können muss die **UIView**, Objekte der Klasse **UIButton** und **UISlider** bilden. Bei einem registrierten Event auf einem Button, führt der Button die entsprechende Methode im verwaltenden Controller der UIView aus. Analog verhält es sich zu den restlichen UIViews innerhalb ihrer ViewController.

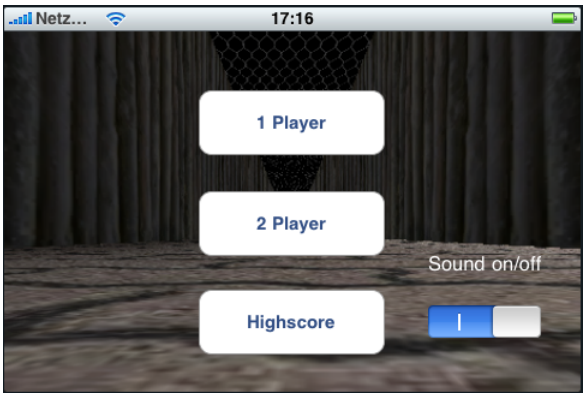

Abbildung 4.24.: Das Hauptmenü

Die Berührungen auf dem Display sollen innerhalb des Spieles Möglichkeiten bieten, den Blickwinkel auf die Spielfigur sowie die Zoomstufe zu ändern.

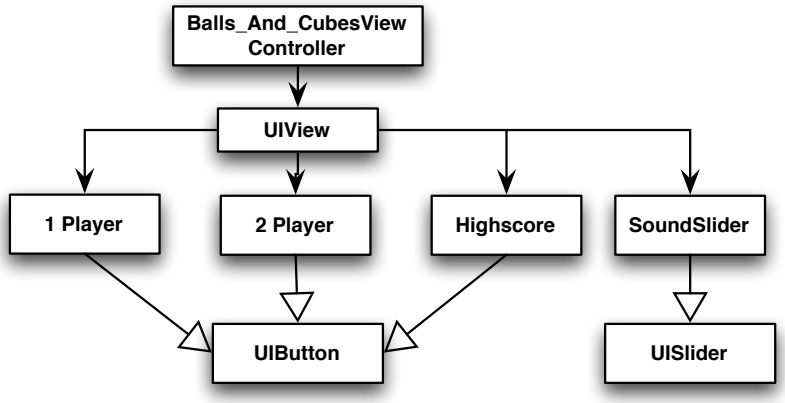

Abbildung 4.25.: Bedienelemente innerhalb einer View

5. Implementierung

Die prototypische Anwendung, die im Rahmen des Systementwurfs beschrieben worden ist, wurde vollständig umgesetzt. Ziel dieses Kapitels wird es sein, die eingesetzten Techniken und Werkzeuge, die zur Umsetzung erforderlich waren genauer zu beschreiben.

5.1. Entwicklungsumgebung

Die Umsetzung der Anwendung auf dem iPhone wurde mit Hilfe der Cocoa API realisiert. Als Programmiersprache dient dazu Objective-C (Vgl. Kapitel 2.5 S. 33). Wie Cocoa aufgebaut ist, kann aus Abbildung 5.1 entnommen werden. In jeder dieser Schichten befinden sich einige Frameworks, die im folgenden kurz dargestellt werden. Core OS enthält unter anderem den Kernel, das Dateisystem, die Netzwerkinfrastruktur sowie die Gerätetreiber. Innerhalb Core Services existieren einige Frameworks, die Stringmanipulationen und den Umgang mit Netzwerken erlauben. Die Media-Schicht ist abhängig von Core Services und bietet der Cocoa Touch Schicht grafische und multimediale Dienste an. In dieser Schicht befindet sich neben Core Animation, Core Audio, Core Graphics auch OpenGL ES. In der Cocoa Touch Schicht befinden sich 2 wichtige Frameworks. Zum einen das UIKit Framework, das die grafische Darstellung und die Registrierung von Events erlaubt. Zum anderen das Foundation Framework, welches das Grundverhalten von Objekten definiert.

Die Entwicklung fürs iPhone geschah mit Hilfe der von Apple kostenlos zur Verfügung gestellten iPhone SDK 2.2.1. Sie enthielt die XCode IDE[64] und den

[64]integrated development environment

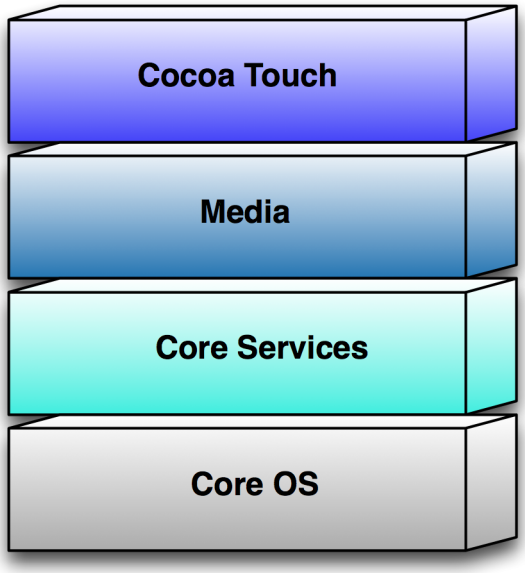

Abbildung 5.1.: Aufbau von Cocoa; Quelle: [7]

Interface Builder, sowie weitere Debugging und Performance Tools. Die eigentliche Programmierung der Anwendung erfolgt innerhalb von XCode, während das Erstellen der Benutzeroberfläche vorwiegend per Drag and Drop im Interface Builder stattfindet. Zum Testen der Anwendung konnte man den iPhone Simulator nutzen, der ebenfalls mit der SDK ausgeliefert wurde. Dieser hatte aber einige Nachteile. Zum einem nahm er die Hardware des zugrunde liegenden Mac's zu Hilfe, wodurch höhere Frameraten mit OpenGL erreicht werden, als diese tatsächlich auf dem iPhone vorkommen, und zum anderen konnte die Sensorik des Beschleunigungssensor nicht getestet werden.

Zum Erstellen der 3D-Objekte wurde das Opensource Modellierungstool Blender in der Version 2.49a verwendet.

5.2. Realisierung des Datenmodells

Die Datenhaltung geschah zweigeteilt. Der Highscore wurde in einer Datenbank auf dem HTW-Server gespeichert. Als DBMS[65] wurde MySQL 5.0 verwendet. Die Administration dieser erfolgte durch phpMyAdmin 2.8.1. Leider

[65]Datenbankmanagementsystem

bietet die Cocoa API keine Methoden um sich mit einer entfernten Datenbank zu verbinden. Infolgedessen musste auf alternative Verfahren zurückgegriffen werden. Für den Zugriff auf die Datenbank in dieser Arbeit wurde eine Kombination aus PHP[66] und JSON[67] verwendet. Im Detail ruft die Klasse **Database** mittels HTTPS[68] ein PHP-Script auf dem HTW-Webserver auf, welches dann durch SQL-Statements die Daten in Form eines JSON-Strings zurückliefert. Um ein gewisses Maß an Sicherheit zu erreichen wurde https anstelle von http verwendet. Folgendes Listing liefert beispielhaft die Top 10 der besten Spieler.

Listing 5.1: PHP Skript zur Ausgabe der Top 10 als JSON String

```
<?
$link = mysql_connect("io.f4.fhtw-berlin.de", "s0514436","");
if ($link!=true){
echo "error";
};
mysql_select_db("s0514436",$link);
$result = mysql_query("SELECT name, points, longitude, latitude FROM highscore ORDER BY
     points DESC LIMIT 10 ;");

for($i = 0; $list[$i] = mysql_fetch_assoc($result); $i++) ;
array_pop($list);
print_r(json_encode($list));
?>
```

Zum Speichern der Einträge in die Datenbank wurden Parameter an das PHP Skript mittels GET übergeben. Der Rückgabewert *YES* oder *NO* wird vom Client genutzt um im Falle eines Fehlers den User zu informieren.

Listing 5.2: PHP Skript zum Eintrag in die Datenbank

```
<?
mysql_select_db("s0514436",$link);
$sql="INSERT INTO highscore (name, points, longitude, latitude) VALUES ('".$_REQUEST['name'].
     "','".$_REQUEST['points']."','".$_REQUEST['longitude']."','".$_REQUEST['latitude']."')";
mysql_query($sql);
if (mysql_affected_rows() == 0)
  echo "NO";
else echo "YES";
?>
```

Der Zugriff auf die PHP Skripte ist in der **Database**-Klasse gekapselt. Aus Abbildung 5.2 können auch die Parameter entnommen werden.

[66] Hypertext Preprocessor
[67] JavaScript Object Notation
[68] Hypertext Transfer Protocol Secure

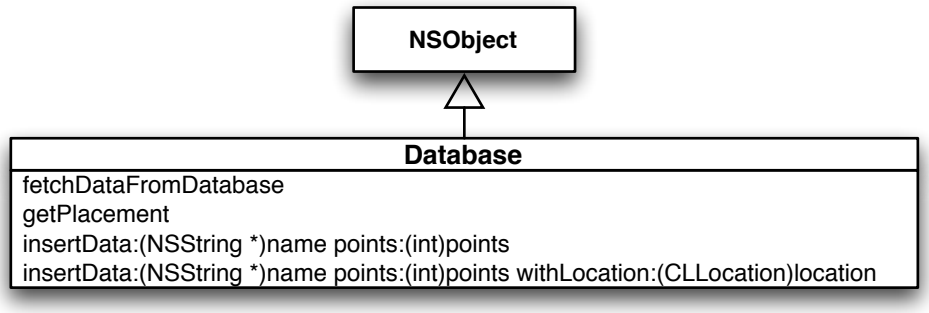

Abbildung 5.2.: Klasse für Datenbankzugriffe

Um die Grundlage für die Einbindung von Objekten in die Objective-C Laufzeitumgebung sicherzustellen muss die Klasse Database von der Root-Klasse **NSObject** abgeleitet sein. Dieses Schema gilt für alle Klassen, sofern sie sich nicht von bereits bestehenden Cocoa Klassen ableiten. Anhand von Listing 5.3 kann beispielhaft die Methodik hinter der Eintragung in die Datenbank eingesehen werden.

Listing 5.3: Methode der Klasse Database für einen Eintrag in die Datenbank

```
1  -(BOOL) insertData:(NSString *)name points:(NSInteger)points withLocation:(CLLocation *)
       location{
2    NSString *latitude = [[NSString alloc] initWithFormat:@"%g", location.coordinate.latitude];
3    NSString *longitude = [[NSString alloc] initWithFormat:@"%g", location.coordinate.longitude
         ];
4    NSString *path = [[NSString alloc] initWithFormat:@"/~s0514436/diplom/addhighscore.php?name
         =%@&points=%d&longitude=%@&latitude=%@",name,points,longitude,latitude];
5    NSURL *theURL = [[NSURL alloc] initWithScheme:@"https"
6                      host:@"www.f4.fhtw-berlin.de"
7                      path:path];
8    NSString *result = [NSString stringWithContentsOfURL:theURL];
9    return [result isEqualToString:@"YES"];
10 }
```

Auf lokaler Ebene wurden Header Dateien, die durch Blender erstellt wurden, für die Beschreibung der 3D Objekte verwendet. Folgendes Listing 5.4 zeigt Auszugsweise die Beschreibung eines Würfels in der **cube.h**. Die Vertexe bilden die einzelnen Punkte im Raum um das Objekt zu beschreiben. Das Array *CubeNormals* findet Verwendung in der Berechnung der Belichtung auf dem Objekt, während die Beschreibung der Texturkoordinaten im Array *CubeTexCoords* stattfindet. Während die Anzahl der Vertexe bei einem Würfel noch sehr gering ist, können diese bei einer Kugel schon viel mehr sein. Durch testen

wurde festgestellt, dass die Kugel als rund empfunden wird, wenn sie durch 960 Vertexe (320 Polygone) beschrieben ist.

Listing 5.4: Auszug aus cube.h für die Beschreibung eines Würfels

```
static const Vertex3D CubeVertices[] = {
{1.000000, 0.999999, 1.000000}, {1.000000, 1.000000, -1.000000}, ...}
static const Vector3D CubeNormals[] = {
{0.333333, 0.500000, 0.166667}, {0.200000, 0.400000, -0.400000}, ...}
static const TextureCoord3D CubeTexCoords[] = {
{1,0}, {1,1},..}
#define kCubeNumberOfVertices 36
#define kCubeNumberOfFaces 12
```

In Abbildung 5.3 kann das 3D-Modell der Objekte betrachtet werden. Hierbei ist anzumerken, das die rechteckigen Flächen des Würfels durch den Export in eine Headerfile in Dreiecke zerlegt werden.

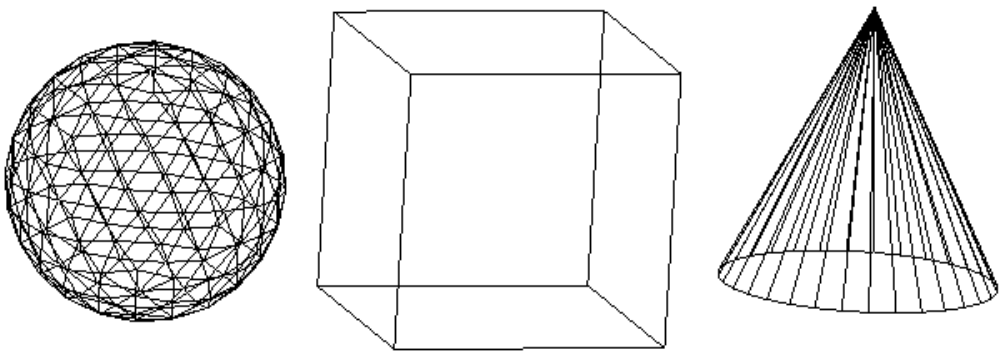

Abbildung 5.3.: Wireframedarstellung der 3D-Modelle

Das Laden der komprimierten Texturen kann beispielhaft aus Listing 5.5 entnommen werden. Die Auflösung der Bilder wurde durch 256*256 Pixel festge-

Listing 5.5: Das Laden von Texturen

```
glGenTextures(7, &texture[0]);
glBindTexture(GL_TEXTURE_2D, texture[i]);
NSString *path = [[NSBundle mainBundle] pathForResource: [pictureArray objectAtIndex:i]
     ofType:@"pvr4"];
NSData *texData = [[NSData alloc] initWithContentsOfFile:path];
glCompressedTexImage2D(GL_TEXTURE_2D, 0, GL_COMPRESSED_RGB_PVRTC_4BPPV1_IMG, 256.0, 256.0, 0,
      (256.0 * 256.0) / 2, [texData bytes]);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
```

legt.

Folgendes Listing 5.6 zeigt auszugsweise wie die Audiodaten in den Speicher geladen werden. Die Klasse **NSBundle** repräsentiert den Ort im Dateisystem wo die die Ressourcen und Klassen des ausgeführten Programmes liegen. Geladen werden die Audiodateien durch die Klasse **SoundEffect**.

Listing 5.6: Das Laden von Audio

```
1 NSBundle *mainBundle = [NSBundle mainBundle];
2 collectSound = [[SoundEffect alloc] initWithContentsOfFile:[mainBundle pathForResource:@"
      chimes" ofType:@"wav"]];
3 dieSound = [[SoundEffect alloc] initWithContentsOfFile:[mainBundle pathForResource:@"hit"
      ofType:@"wav"]];
```

5.3. Realisierung der Funktionalität

Die meiste Funktionalität in der Anwendungslogik stellt die Kommunikation und die Auswertung der Sensordaten da. Folglich wird die Implementierung dieser detaillierter dargestellt.

5.3.0.1. Kommunikation mit dem Spielgegner

Die Kommunikation zwischen den Geräten gestaltete sich schwieriger als erwartet. Im Internet findet man einige fertige frei benutzbare Klassen die eine Abstraktion des Datenaustausches auf Basis von UDP vornehmen. Aber Tests haben gezeigt das diese nicht den Realtime Anforderungen meines Spiels gewachsen waren. Dadurch wurden konzeptuell eigene Klassen entwickelt (Vgl. Kapitel 4.3.3 S.70.

Für den Mac gibt es eine Technologie namens *Distributed Objects*, die es sehr einfach macht mit Prozessen auf anderen Rechnern zu kommunizieren. Leider funktioniert diese Technik nur von Mac zu Mac. Trotzdem gibt es fürs iPhone einige verschiedene Möglichkeiten der Umsetzung. Apple bietet das C-basierte **CFNetwork**-Framework und das Objective-C basierte **Foundation**-Framework zur Socket-Kommunikation an. Da diese Frameworks ein gewisses

Neuland bergen, kann auch auf die low-level BSD[69]-Netzwerk API zurückgegriffen werden. Was im Falle dieser Arbeit getan wurde.

Die low-level Funktionalität wurde in der Klasse **Socket** implementiert. Listing 5.7 zeigt die wichtigsten Funktionen, die innerhalb der Socket Klasse zu finden sind. Die Struktur sockaddr_in beinhaltet die Protokollfamilie, die

Listing 5.7: wichtige Funktionen des Socket Klasse

```
1  //create the socket
2  self._socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
3  //bind the socket
4  struct sockaddr_in address;
5  address.sin_family = AF_INET;
6  address.sin_addr.s_addr = INADDR_ANY;
7  address.sin_port = htons((unsigned short) port);
8  bind(self._socket, (struct sockaddr*) &address, sizeof(address));
9  //set non-blocking
10 fcntl(self._socket, F_SETFL, O_NONBLOCK, nonBlocking)
11 //send data
12 struct sockaddr_in *address = (struct sockaddr_in *)[destination socketAddress];
13 int size = address->sin_len;
14 int sentBytes = sendto(self._socket, [data bytes], [data length], 0, (const struct sockaddr*)
       address, size);
15 //receive data
16 struct sockaddr_in from;
17 socklen_t fromLength = sizeof(from);
18 int receivedBytes = recvfrom(self._socket, buffer, 512, 0, (struct sockaddr*)&from, &
      fromLength);
```

IP-Adresse und den Port. Funktionen wie bind und recvfrom greifen dann auf diese Struktur zu.

Diese low-level Funktionalität wird von der Klasse **Connection** verborgen. Das folgende Listing 5.8 zeigt wie die Klasse Connection den Paketversand realisiert. Der zu versendende Payload kommt in Form eines NSData Objektes

Listing 5.8: Packet senden durch die Klasse Connection

```
1  -(BOOL) sendPacket:(NSData *)data withType:(NSInteger)_type{
2    assert(self.running);
3    if (self.address == nil) return NO;
4    NSTimeInterval time = [NSDate timeIntervalSinceReferenceDate];
5    Packet *p = [[Packet alloc] initWithData:data type:_type sequenz:sequenz ack:remotesequenz
         protocol:protocolId sendTime:(double)time];
6    NSData *sendArchive = [NSKeyedArchiver archivedDataWithRootObject:p];  [p release];
7    //increase sequenz nr
8    sequenz ++;
9    return [self.socket sendData:sendArchive To:[self address]];
10 }
```

[69]Berkeley Software Distribution

(ein Byte Buffer) an. Sofern die Adresse des Ziels bekannt ist und eine Verbindung steht, wird ein Objekt der Klasse **Packet** mit sämtlichen Informationen (Sequenznummer, Ack, Payload, gesendete Zeit) erzeugt. Dieses Objekt wird durch einen **NSKeyedArchiver** serialisiert um dann zur **sendData** Methode der Socketinstanz geschickt zu werden. Um serialisiert werden zu können muss die Klasse Packet das Protokoll **NSCoding** implementieren. Die Methoden im Listing 5.9 nehmen das Kodieren und Dekodieren vor.

Listing 5.9: Serialisierungsmethoden des Protokoll NSCoding

```
1 -(id)initWithCoder:(NSCoder *)coder {...}
2 -(void)encodeWithCoder:(NSCoder *)coder {..}
```

Der Client und der Host versuchen in jeden Frame die Positionsdaten des anderen Spielers zu empfangen. Der Client weiß aber nicht wann er Statusänderungen und Angaben zu den Spielobjekten empfangen soll. Aus diesem Grund wurden die Pakete mit Typnummern versehen. Sobald ein Paket mit der Typnummer 4 eintrifft weiß der Client, dass es sich um ein Spielobjekt handelt. Bei Typ 5 handelt es sich um Statusinformationen. Die Methode *receivePacketWithType:* der Klasse **Connection** teilt mittels Delegates den **GameController** mit, dass Informationen zu Spielobjekten und Statusinformationen eingetroffen sind.

Listing 5.10: Delegates zur Benachrichtigung des GameControllers

```
1  //enemy received
2  if ([packet type] == 4){
3    Enemy *enemy = [[Enemy alloc] buildEnemyFromData:[packet payload]];
4    [self.delegate connectionReceivedEnemy:enemy];
5    return nil;
6  }
7  // status received
8  if ([packet type] == 5){
9    NSString *status = [[NSString alloc] initWithData:[packet payload] encoding:1];
10   [self.delegate connectionReceivedStatus:status];
11   return nil;
12 }
```

Damit Bonjour funktionieren kann und die Latenz gering ist, wurde die Nutzung auf das WLAN begrenzt. Um festzustellen ob eine WLAN Verbindung anliegt, wurde die Klasse **Reachability** implementiert. Zudem wurde eine Variable *localWiFiConnectionStatus* vom Typ *NetworkStatus* innerhalb der Klasse **Balls_And_CubesViewController** (Hauptmenü) deklariert. Sie enthält

den Status einer Wi-Fi Verbindung. Um diese Variable mit dem Status zu füllen, muss der Balls_And_CubesViewController als Observer bei der Instanz des **NSNotificationCenters** registriert werden. Sofern sich die Empfangsbedingung ändert, wird die Methode *reachabilityChanged* im Controller aufgerufen, die den Status der Verbindung in der Reachability Klasse abfragt. Dies aktualisiert den Verbindungsstatus. Listing 5.11 zeigt die Registrierung, die Aktualisierung und die Überprüfung des Status.

Listing 5.11: Überprüfung des Verbindungsstatus

```
1 //register
2 [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(reachabilityChanged
      :)
3 name:@"kNetworkReachabilityChangedNotification" object:nil];
4 //update status
5 self.localWiFiConnectionStatus = [[Reachability sharedReachability] localWiFiConnectionStatus
      ];
6 //check status
7 if (self.localWiFiConnectionStatus == ReachableViaWiFiNetwork) {/*start Game*/}
```

Das veröffentlichen eines Dienstes mittels Bonjour geht leicht innerhalb des **HostController** von statten. Aus Listing 5.12 kann ersehen werden, das zuerst ein Socket geöffnet werden muss und anschließend ein Dienst durch eine Instanz des **NSNetservice** veröffentlicht wird. Jetzt wartet der Host bis

Listing 5.12: Veröffentlichen eines Dienstes am Host mit Bonjour

```
1 //create socket
2 self.connection = [[Connection alloc] initConnectionWithProtocolId:99887766 Timeout:5.0];
3 [self.connection startConnection:port])
4 //create the service
5 netService =
6    [[NSNetService alloc] initWithDomain:@"" type:@"_balls._udp." name:@"" port:port];
7 netService.delegate = self;
8 //publish the netservice on port
9 [netService publish];
```

er eine Nachricht vom Client bekommt. Wenn eine Nachricht eingetroffen ist muss er den **Balls_And_CubesViewController** mitteilen, dass er einen Spielpartner gefunden hat, denn dieser ist in der Lage das Spiel zu starten. Dies funktioniert mit Delegates und Protokollen. Delegates sind Objekte, die einen Verweis auf eine Methode besitzen. Mit ihnen ist es möglich innerhalb einer Klasse, Nachrichten an andere Klassen zu schicken. Der HostController abstrahiert die Protokollmethode *hostController: DidFindOpponent:*, welche

im Balls_And_CubesViewController implementiert wird. Durch den Aufruf des Delegates im HostController wird die Methode im Balls_And_CubesViewController ausgeführt (Vgl. Listing 5.13). Durch die Weitergabe der In-

Listing 5.13: Benachrichtigung durch Delegates

```
1 [self.delegate hostController:self DidFindOpponent:self.connection];
```

stanz *self.connection* kann der Balls_And_CubesViewController das Spiel mit eben dieser als Host erstellen. Prinzipiell ist das die gleiche Vorgehensweise auf Clientseite. Aus diesem Grund wird hier auf eine Darstellung verzichtet.

5.3.0.2. Auswertung der Sensoren

Eines der wichtigsten Elemente im Spiel ist die Auswertung des Beschleunigungssensor. Dies gestaltet sich relative einfach. Die Auswertung findet innerhalb des für die Spiellogik zuständigen **GameController** statt. Dieser implementiert das Protokoll **UIAccelerometerDelegate** und die Methode *accelerometer: didAccelerate:*. Um für einen Aufruf dieser Methode zu sorgen, muss der Accelerometer, der bereits jeder Anwendung zur Verfügung steht, konfiguriert werden (Vgl. Listing 5.14). Mit *setUpdateInterval* wird bestimmt

Listing 5.14: Konfiguration des Accelerometers

```
1 [[UIAccelerometer sharedAccelerometer] setUpdateInterval:(1.0 / 60)];
2 [[UIAccelerometer sharedAccelerometer] setDelegate:self];
```

wie oft der Beschleunigungssensor ausgelesen werden soll. So wird das Auslesen des Sensors zum aktuellen *Runloop* hinzugefügt. Der *RunLoop* ist eine Programmschleife, die im Mainthread des Programmes ausgeführt wird, und automatisch beim Start erzeugt wurde. Des weiteren muss der Delegate auf die Klasse gesetzt werden, die die Auswertung vornimmt. Indem Fall der **GameController**. Für die Steuerung der Spielfigur ist der Beschleunigungswert entlang der Y-Achse des iPhones interessant. Die Y-Achse weil wir das Spiel im Breitbildformat spielen. Um ein ruckeln der Spielfigur, obwohl sie sich nicht bewegt, zu vermeiden muss ein Tiefpassfilter auf die Sensordaten angewandt werden. Dadurch werden kurze Ausschläge in der Werten vermieden und die

Spielfigur spiegelt den Eindruck einer sanften Bewegung wieder. Der resultierende Wert wird als Geschwindigkeit der Spielfigur interpretiert (Vgl. Listing 5.15).

Listing 5.15: Auswertung des Accelerometers

```
- (void)accelerometer:(UIAccelerometer*)accelerometer didAccelerate:(UIAcceleration*)
    acceleration{
  //kFilteringFactor =  0.05
  accel[1] = acceleration.y * kFilteringFactor + accel[1] * (1.0 - kFilteringFactor);
  [self.player setVelocity:-accel[1]];
}
```

Zur Auswertung der Berührungen auf dem Display spielt der **Responder Chain** eine wichtige Rolle. Der Responder Chain stellt eine Weiterleitungskette von Events dar. Das oberste Glied, der **First Responder** dieser Kette, ist die grafische Darstellung der OpenGL Objekte in der **glView**. Diese Instanz der **UIView** nimmt die Eingaben entgegen. Falls keine Auswertungen der Berührungen in der View stattfinden, werden sie an den verwaltenden Controller (GameController), der **glView** weitergeleitet. Diese Weiterleitung funktioniert weil die Klassen UIView und UIViewController vom **UIResponder** abgeleitet sind. Um letztendlich die Auswertung der Daten vorzunehmen, werden Methoden aus der Klasse UIResponder im GameController implementiert. Durch die Methode *touchesBegan: withEvent:* ist es möglich durch zählen der Taps die Kameraposition zurückzusetzen. Listing 5.16 zeigt wie bei 2 maligen tippen die Kameraposition in Ausgangsposition zurückgesetzt wird. Um den Blickwinkel

Listing 5.16: Rücksetzen der Kameraposition

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
  UITouch *touch = [touches anyObject];
  NSUInteger tapCount = [touch tapCount];
  switch (tapCount) {
  case 2:
     //reset camera
     cameraPosition.x = 0.0f; cameraPosition.y = 1.5f; cameraPosition.z = 2.0f;
     break;
  }
}
```

und die Zoomstufe auf das Spielfeld zu verändern muss zuerst die **glView** für Multitouch konfiguriert werden. Anhand des Objektes *touches* lässt sich in der Methode *touchesMoved: withEvent:* ermitteln, wie viele Berührungen auf dem

Display zu erkennen sind. Bei einer Berührung wird lediglich der Blickwinkel auf die Spielfigur verändert, während bei 2 Berührungen und die Auswertung der Bewegung die Zoomstufe geändert wird. Dies geschieht indem die Differenz der Distanz der 2 Berührungspunkte vor und nach der Bewegung zur Änderung der Kameraposition beiträgt. Listing 5.17 zeigt dieses.

Listing 5.17: Verändern der Kameraposition

```
1  //enable multitouch
2  [self.glView setMultipleTouchEnabled:YES];
3  - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
4    UITouch *touchA, *touchB;
5    CGPoint pointA, pointB;
6    if ([touches count] == 1) {
7      touchA = [[touches allObjects] objectAtIndex:0];
8      pointA = [touchA locationInView:glView];
9      pointB = [touchA previousLocationInView:glView];
10     float disty = pointA.y-pointB.y;
11     float distx = pointB.x- pointA.x;
12     cameraPosition.y += 0.05 * disty;
13     cameraPosition.x += 0.05 * distx;
14   }
15   if ([touches count] == 2) {
16     touchA = [[touches allObjects] objectAtIndex:0];
17     touchB = [[touches allObjects] objectAtIndex:1];
18     pointA = [touchA locationInView:glView];
19     pointB = [touchB locationInView:glView];
20     float currDistance = [self distanceFromPoint:pointA toPoint:pointB];
21     pointA = [touchA previousLocationInView:glView];
22     pointB = [touchB previousLocationInView:glView];
23     float prevDistance = [self distanceFromPoint:pointA toPoint:pointB];
24     cameraPosition.z += 0.005 * (currDistance - prevDistance);
25   }
26 }
```

Die Ermittlung der Positionsdaten spielt innerhalb des Spiels keine Rolle, vielmehr sind sie ein netter Zusatz für den Highscore. Wie im Konzept beschrieben implementiert die **AddHighscore**-Klasse ein Objekt der **LocationController**-Klasse. Durch das Protokoll **LocationControllerDelegate** und einem Delegate im LocationController kann das AddHighscore-Objekt benachrichtigt werden, wenn eine Ortung stattgefunden hat. Um eine Ortung durchführen zu können muss das **CoreLocation Framework** im LocationController eingebunden werden. Weiterhin muss eine Instanz des **CLLocationManager** im Controller konfiguriert werden. Da die Genauigkeit der Ortung keine große Rolle spielt, reicht es die Konstante *kCLLocationAccuracyBest* zu verwenden. Das iPhone verwendet mehrere Technologie zur Positionsbestimmung (Vgl. Kapitel 2.2.1 S. 5). Durch diese Konstante wird die Ortung anhand der zur Zeit bestmöglichen Technik vorgenommen. Da die Positionsbestimmung sehr

viel Batteriekapazität verbraucht muss die Lokalisierung explizit gestartet und beendet werden.

Listing 5.18: Initialisierung zur Positionsbestimmung

```
1 //init
2 locationManager = [[CLLocationManager alloc] init];
3 locationManager.delegate = self;
4 //set accuracy
5 locationManager.desiredAccuracy = kCLLocationAccuracyBest;
6 //start updating location
7 [locationManager startUpdatingLocation];
```

5.3.0.3. allgemeine Funktionalität im Spiel

Die Klasse **GameController** ist die wichtigste Klasse während des Spielablaufs. Sie kapselt die ganze Logik, die zur Darstellung und Ausführung des Spielgeschehens notwendig ist. In Abbildung 5.4 sind die wichtigsten Instanzvariablen und Methoden zu erkennen.

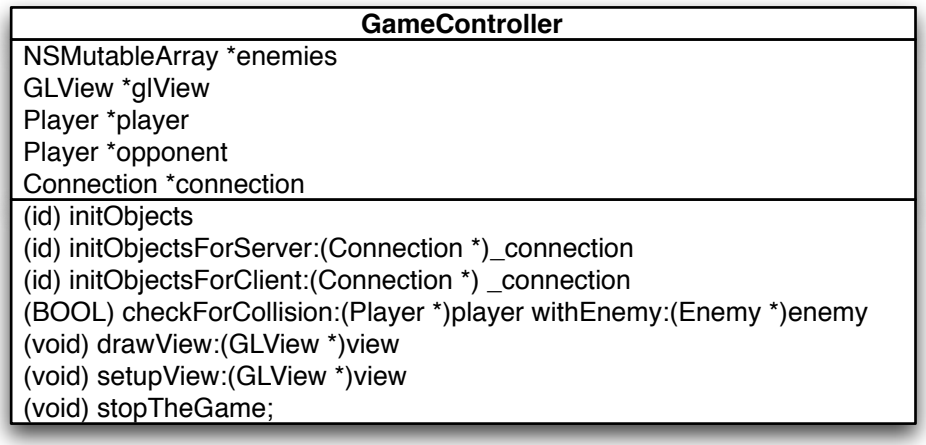

Abbildung 5.4.: Klassendiagramm des GameControllers

Die Methode *setupView* konfiguriert die verschiedenen Parameter die zur Darstellung der Spielinhalte von Bedeutung sind. So werden in der Funktion verschiedene Angaben zum Licht gemacht. Außerdem werden die Materialeigenschaften, die Perspektive und diverse Rendereinstellungen getätigt. Leider ist die *glut*-Bibliothek auf dem iPhone nicht zu verwenden. So können die wichtigen Funktionen *gluPerspective* und *gluLookAt* nicht verwendet werden. Abhilfe

wurde durch frei verfügbare Funktionen geschaffen, die diese Funktionalität implementieren. Die beiden Funktionen sind im Anhang S. VII zu sehen.

Da es sich bei einem iPhone um ein Telefon handelt, muss sichergestellt werden, dass das Spiel pausiert sofern man einen Anruf bekommt. Um dies zu gewährleisten implementiert der **GameController** das Protokoll **UIApplicationDelegate**. Sofern das Gerät einen Anruf registriert ruft ein Delegate die Methode *applicationWillResignActive:* auf. Innerhalb dieser Methode wird die Animation der **glView** gestoppt. Nach Beendigung des Anrufs, wird die gegenteilige Methode *applicationDidBecomeActive* aufgerufen, welche die Animation wieder fortsetzt. Listing 5.19 zeigt dies.

Listing 5.19: Aktivität der Anwendung

```
1 - (void)applicationWillResignActive:(UIApplication *)application{
2   [self.glView stopAnimation];
3 }
4 - (void)applicationDidBecomeActive:(UIApplication *)application{
5   [self.glView startAnimation];
6 }
```

Außerdem versucht das iPhone Strom zu sparen indem es bei längerer Inaktivität (keine Berührung des Displays) den Bildschirm dimmt. Um dies zu unterbinden muss bei der Initialisierung des GameControllers, der in Listing 5.20 stehende Code eingebunden werden.

Listing 5.20: Deaktivierung des Dimmers

```
1 if (![[UIApplication sharedApplication] isIdleTimerDisabled])
2   [[UIApplication sharedApplication] setIdleTimerDisabled:YES];
```

Um die Position eines aufgestellten Highscores zu sehen, wird das bereits vorinstallierte Programm *Karten* verwendet. Das hat zur Folge, dass das Spiel beendet wird und *Karten* geöffnet wird, da nur eine Applikation zur selben Zeit laufen kann. Durch die Instanz der **UIApplication**-Klasse kann mittels *openURL* ein Systemcall ausgeführt werden. Durch die URL, sichtbar in Listing 5.21, weiß die UIApplication Instanz das nicht der Webbrowser, sondern die entsprechende Anwendung gestartet werden muss.

Listing 5.21: Systemcall um Karten zu öffnen

```
1 NSString *url = [NSString stringWithFormat:@"http://maps.google.com/maps?ll=%@",loc];
2 [[UIApplication sharedApplication] openURL:[NSURL URLWithString:url]];
```

Im Entwurf hieß es, dass ein geeigneter Pseudozufallsgenerator zur Initialisierung der Spielobjekte genutzt werden muss. Der **arc4random()** Zufallsalgorithmus liefert annehmbare Werte. Zudem hat er die doppelte Reichweite als der vergleichbare **rand()**. Der Maximalwert der durch arc4random auf dem iPhone erreicht werden kann beträgt 4.294.967.296. Aus dem folgenden Listing kann entnommen werden, dass die zufällige Position eines Spielobjektes durch eine Zufallszahl zwischen 0 und 3800 ermittelt wird. Subtrahiert man von dieser 1900 und teilt das Ergebnis durch 1000, erhält man Werte zwischen -1.9 und +1.9. Was zuzüglich des Radius eines Spielobjektes genau den Rahmen der Spielfläche wiedergibt. Der Typ eines Spielobjektes wird durch eine

Listing 5.22: Zufallszahlen

```
1 #define ARC4RANDOM_MAX 0x100000000
2 //random position
3 GLfloat _x = (floorf(((GLfloat)arc4random() / ARC4RANDOM_MAX) * 3800.0) - 1900.0) / 1000;
4 //random type
5 double type = floorf(((double)arc4random() / ARC4RANDOM_MAX) * 100.0);
```

Zufallszahl zwischen 0 und 100 bestimmt. Da die Typen der Spielobjekte in unterschiedlichen Häufigkeiten auftreten sollen gilt folgende Aufteilung:

- type >= 95 : Überraschungspaket
- type < 95 und type >= 85 : Kegel
- type < 85 und type >= 55 : Totenkopf
- type < 55 : normale Kugel

Ähnlich funktionieren die Zufälligkeit bei der Bestimmung der Startposition und der Geschwindigkeit.

Spielobjekte werden nach jedem Frame gesetzt. Durch Schwankungen in der Frameanzahl pro Sekunde und um generell eine weichere Bewegung der Objekte darzustellen wird das Setzen der Objekte abhängig von der Zeit, die seit

dem letzten Zeichnen vergangen ist, gesetzt. Die Geschwindigkeit (Geschwindigkeit + SpeedUp + aktuelles Level * 2) der einzelnen Spielobjekte wird mit der Zeitdifferenz multipliziert. Listing 5.23 demonstriert dies. Analog verhält es sich mit der Spielerfigur. Zudem wird auch der Rotationswinkel der Spielfigur

Listing 5.23: Setzen der Spielobjekte abhängig von der verstrichenen Zeit

```
1  if (lastDraw) {
2    NSTimeInterval timeSinceLastDraw = [NSDate timeIntervalSinceReferenceDate] - lastDraw;
3    z_pos += (self.velocity + speedup +(GLfloat)_level*2.0 ) * timeSinceLastDraw;
4  }
5  lastDraw = [NSDate timeIntervalSinceReferenceDate];
```

abhängig von der verstrichenen Zeit und der Geschwindigkeit ermittelt.

Die Kollisionsüberprüfung wird mittels eines einfachen Sphere-Sphere Algorithmus vorgenommen. Also wenn der Radius des Spielers + der Radius eines Spielobjektes größer-gleich der euklidischen Distanz zwischen beiden ist, dann ist eine Kollision vorhanden. Es ist nur von Bedeutung ob eine Kollision stattfindet. Aussagen über den genauen Punkt der Kollision und der Geschwindigkeit mit der dieser passiert ist nicht wichtig. Bei hohen Geschwindigkeiten der Spielobjekte kann der sogenannte Tunneleffekt auftreten (Vgl. Abbildung 5.5). Das heißt, dass der Spieler in dem Fall vom Spielobjekt übersprungen

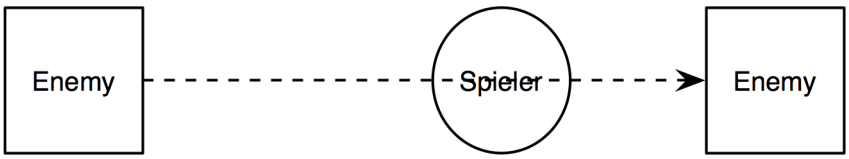

Abbildung 5.5.: Tunneleffekt bei Kollisionserkennung

wird. Hier gibt es viele komplexe Algorithmen um dieses zu vermeiden. Der Ansatz, die Kollisionsüberprüfung noch auf die letzte Position des Spielobjektes zu erweitern reicht in dem Fall aus. Es wird zusätzlich überprüft ob der Spieler sich zwischen aktueller und letzter Position des Spielobjektes befand. Weiterhin startet die Kollisionsüberprüfung erst wenn die Spielobjekte in die Nähe der Spielfigur kommen. Sie setzt erst bei einem Z-Wert von -8 ein. Diese Methodik ist auch nicht perfekt, aber für den Anwendungsfall vollkommen ausreichend. Der Tunneleffekt tritt zudem erst in den höheren Level auf. Listing 5.24 beschreibt die Kollisionsabfrage innerhalb des **GameController**.

Listing 5.24: Kollisionsabfrage

```
GLfloat xD = fabs(aPlayer.x_pos - aEnemy.x_pos);
GLfloat zD = fabs(aPlayer.z_pos - aEnemy.z_pos);
GLfloat distPlayerEnemy = sqrt(xD*xD+zD*zD);
GLfloat distRect = fabs(aPlayer.x_pos - aEnemy.x_poslast) - (aPlayer.size + aEnemy.size);
if ((distPlayerEnemy <= (aPlayer.size + aEnemy.size)) ||
    (distRect <= 0 && aPlayer.z_pos < aEnemy.z_pos && aPlayer.z_pos > aEnemy.z_poslast)){
    //collision
}
```

5.4. Realisierung der Präsentation

Zur Umsetzung der View bietet Apple, wie bereits erwähnt, den Interface Builder an. Durch einfaches Drag and Drop werden die UI-Elemente einfach miteinander verknüpft. So wurden auch in dieser Arbeit die meisten Views mit ihren Bedienelementen erstellt. Der Interface Builder speichert diese Ressourcen in sogenannte NIB-Dateien. Diese Dateien können im Interface Builder direkt mit UIViewControllern verbunden werden. Eine Auswahl der vorgefertigten UI-Elemente kann aus Abbildung 5.6 entnommen werden. Sofern man

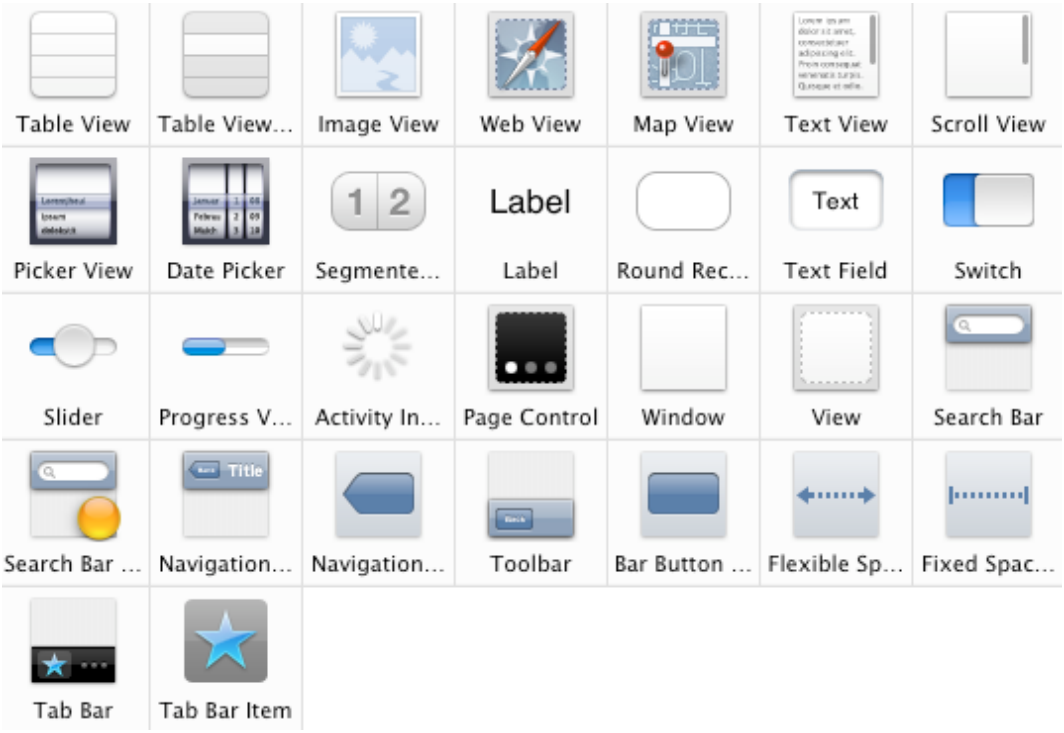

Abbildung 5.6.: Userinterface Objekte

Eigenschaften von UI-Objekte abfragen oder verändern will, müssen diese in-

nerhalb des verwaltenden ViewControllers mit *IBOutlet* deklariert werden. Beispielsweise sei dies anhand des Switchs für die Soundausgabe im Hauptmenü (Balls_And_CubesViewController) dargestellt: `IBOutlet UISwitch *soundSwitcher ;`. Für die Buttons im Hauptmenü muss dieses nicht vollzogen werden, da kein Bedarf darin besteht die Eigenschaften zu ändern oder auszulesen. Vielmehr werden diese Buttons mit Methoden in den ViewControllern verknüpft. Daher müssen solche Methoden als Rückgabewert *IBAction* (deklariert als void) haben. Das Kürzel IB steht für den Interface Builder. Das Verknüpfen der Buttons mit auszuführenden Methoden funktioniert ebenfalls per Drag and Drop. Als Auslöser für den Aufruf einer Methode durch einen Button existieren verschiedene Events. Es muss definiert werden nach welchen Event die Methode aufgerufen werden soll. Das Event *Touch Up Inside* bildet hierbei den häufigsten Anwendungsfall.

Das bedeutet, dass die Methode bei Berührung eines Buttons erst dann aufgerufen wird wenn das Loslassen dessen auch innerhalb des Buttons erfolgt. Demzufolge würde *Touch Up Outside* eine Methode aufrufen, sofern die Berührung innerhalb und das Loslassen außerhalb eines Buttons erfolgt. Abbildung 5.7 zeigt verschiedene Events die auf einen Button registriert werden können.

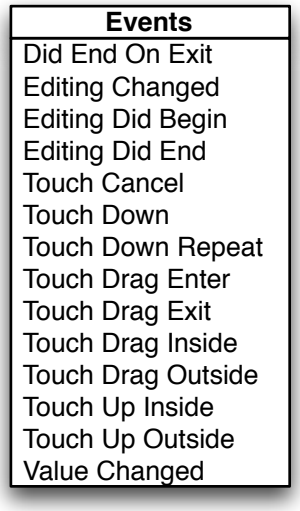

Abbildung 5.7.: Events

Der Aufbau der Benutzeroberfläche innerhalb des Spiels wurde programmiertechnisch umgesetzt. Die View zur Anzeige der OpenGL Welt muss zuvor auf den OpenGL Kontext gesetzt werden. Die Initialisierung dessen kann aus Listing 5.25 entnommen werden. Hierbei wird durch den *animationTimer* festgelegt wie oft der Runloop der Anwendung versuchen soll die OpenGL Welt in einer Sekunde darzustellen. Durch die Methoden *startAnimation* und *stopAnimation* wird die Darstellung dessen gestartet bzw. pausiert. Ein Anwendungsfall dessen wurde bei ankommenden Anrufen in Kapitel 5.3.0.3 S.94 beschrieben.

Listing 5.25: Initialisierung der OpenGL View

```
-(id)initGLES{
  CAEAGLLayer *eaglLayer = (CAEAGLLayer*) self.layer;
  // Configure it so that it is opaque, does not retain the contents of the backbuffer when
       displayed, and uses RGBA8888 color.
  eaglLayer.opaque = YES;
  eaglLayer.drawableProperties = [NSDictionary dictionaryWithObjectsAndKeys:
                  [NSNumber numberWithBool:FALSE], kEAGLDrawablePropertyRetainedBacking,
                  kEAGLColorFormatRGBA8, kEAGLDrawablePropertyColorFormat,
                  nil];
  // Create the EAGLContext, and if successful make it current and create the framebuffer.
  context = [[EAGLContext alloc] initWithAPI:kEAGLRenderingAPIOpenGLES1];
  if(!context || ![EAGLContext setCurrentContext:context] || ![self createFramebuffer]){
    [self release];
    return nil;
  }
  // Default the animation interval to 1/60th of a second.
  animationInterval = 1.0 / kRenderingFrequency;
  return self;
}
- (void)startAnimation{
  animationTimer =
    [NSTimer scheduledTimerWithTimeInterval:animationInterval target:self selector:@selector(
         drawView) userInfo:nil repeats:YES];
}
- (void)stopAnimation{
  [animationTimer invalidate];
  animationTimer = nil;
}
```

Im Entwurf wurde ein NavigationController erwähnt, der dafür zuständig ist die einzelnen ViewController miteinander in Verbindung zu setzen um eine einfache Navigation zu ermöglichen. Ferner kann festgelegt werden ob der Übergang zwischen den Controllern animiert stattfinden soll oder nicht. Listing 5.26 zeigt die entsprechenden Aufrufe dazu.

Listing 5.26: Nutzung eines UINavigationControllers

```
//Aufruf im Balls_And_CubesViewController zum Start des Spiels
[self.navigationController presentModalViewController:gameController animated:YES];
//Aufruf im GameController um zum Hauptmenue zurueckzukehren
[self.navigationController dismissModalViewControllerAnimated:YES];
```

6. Evaluation und Demonstration

Im folgenden Kapitel wird eine Bewertung und Demonstration des Spiels stattfinden.

6.1. Bewertung der Anwendung

Die Bewertung bei Spielen gestaltet sich immer relativ schwierig was den Spielspaß angeht. Aus diesem Grund werden Bewertungen anhand der von dem Beschleunigungssensor ermittelten Werte und der Zeit, die zum Übertragen der Pakete gebraucht wird, abgegeben. Außerdem werden Angaben zu den erreichten Frames pro Sekunde in Abhängigkeit darzustellender Polygone gemacht.

Um auf dem iPhone testen zu können, ist es erforderlich sich bei Apple für das Developer Programm zu registrieren. Diese Registrierung kostet jährlich 99$. Sonst gibt es nur die Möglichkeit die Anwendung im iPhone Simulator zu überprüfen. Wie bereits erwähnt verwendet dieser Simulator die zugrunde liegende Hardware zur Ausführung der Programme. Aus diesem Grund würden die Messergebnisse keine korrekten Werte widerspiegeln.

Da der Spielspaß oder generell die Ausführung stark von der Framerate abhängig ist, sollen diese im ersten Schritt untersucht werden. In der Implementierung wurde die Rate zur Aktualisierung des Bildes auf 1/60 der Sekunde bestimmt. Dieser Wert von 60 Frames pro Sekunde wurde im Simulator auch erreicht. Auf dem iPhone hingegen lag er ca. bei der Hälfte. Aus Abbildung 6.1 kann der Verlauf der Framerate während eines Solospiels entnommen werden. Im absoluten Durchschnitt wurden 32 fps[70] erreicht. Weiterhin ist anhand der Durchschnittslinie erkennbar, dass die Frameanzahl mit steigenden Spielfortschritt sinkt. Zu begründen ist dies durch die Zunahme der darzustellenden

[70] Frames per Second

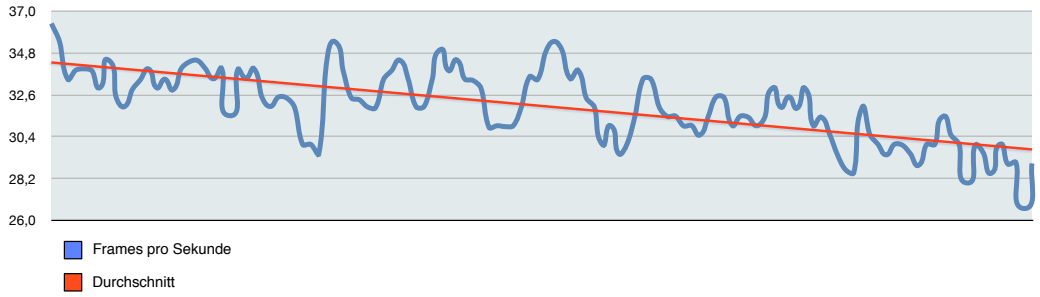

Abbildung 6.1.: Frames pro Sekunde im Solospiel

Objekte, die mit Erhöhung des Levels einhergehen. Während im ersten Level nur der Spieler, das Spielfeld und zwei Spielobjekte zu zeichnen sind, liegt die maximale Zahl der Objekte in Level 8 schon bei 10. Außerdem muss die Spiellogik in Form von Kollisionserkennung nun auf mehrere Objekte angewandt werden.

Im Netzwerkspiel dagegen ist eine Verminderung der Frames pro Sekunde von im Durchschnitt 5 Frames zu erkennen (Vgl. Abbildung 6.2). Auch hier entsteht der gleiche Verlauf, wie im Solospiel. Der allgemeine Unterschied von 5 Frames lässt sich auf den Austausch des Gamestates, sowie die Anwendung der Kollisionserkennung auf 2 Spieler zurückführen. Trotzdem werden im absoluten immer noch 27 Frames pro Sekunde erreicht. Anhand dieser Zahlen kann

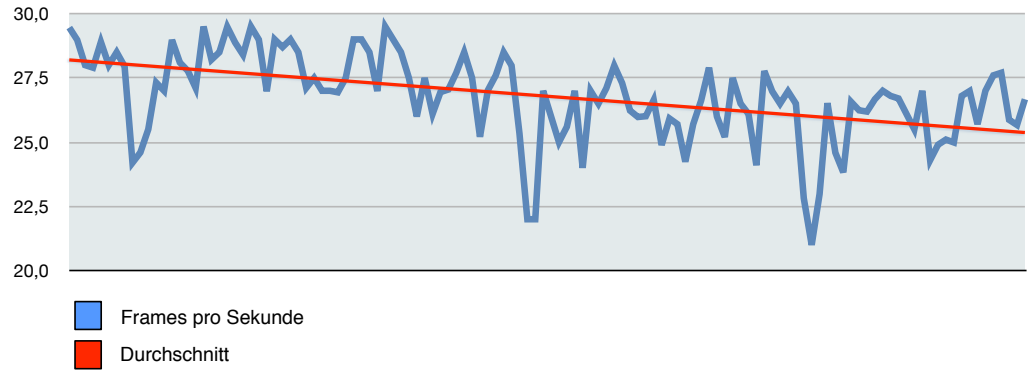

Abbildung 6.2.: Frames pro Sekunde im Netzwerkspiel

das Spielgeschehen als flüssig bewertet werden.

Als nächstes wird die Latenz einer Betrachtung unterzogen, da sie der maßgebliche Faktor ist um die Fairness im Netzwerkspiel sicherzustellen. Als Basis zum Test wurde ein herkömmlicher Standard Router von Vodafone (Easybox A 600) verwendet. Im ersten Testdurchlauf wurde die Zeit gemessen, die ein Paket vom Client zum Host und wieder zurück braucht. Die Latenz spiegelt die Hälfte dieses Messwertes wieder. Es wurde außerhalb der Spiellogik jede Sekunde ein Paket verschickt und gewartet bis es wieder ankommt. Die ermittelten Werte können in Abbildung 6.3 anhand der blauen Linie entnommen werden. Im Durchschnitt lag die Latenz bei 2 ms. Dagegen kann im Spielverlauf bei einem Paketversand entsprechend der Framerate eine Latenz bis zu 15 ms entstehen (grüne Linie). Die aber sofern man die Latenzen mit Spielen im Internet vergleicht, wo Werte kaum unter 60 ms fallen, immer noch einen guten Wert darstellt. Beeindruckend ist auch die Zuverlässigkeit von UDP im lokalen

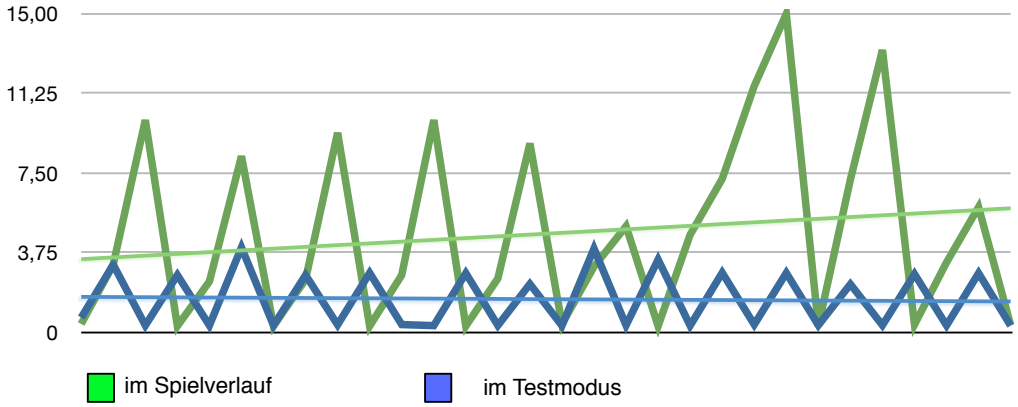

Abbildung 6.3.: Latenzen im Vergleich

Netz. Nur selten konnte ein verlorenes Paket oder eins außerhalb der Reihenfolge erkannt werden. Weiterhin wurde getestet ob durch die Verwendung von mehreren Access Points[71] zum WLAN eine Veränderung in der Latenz ersichtlich wurde. Die Unterschiede waren so marginal das diese nicht weiter erwähnt werden müssen. Auch der Bonjour Dienst funktioniert tadellos solange wie sich die Kommunikationspartner im selben IP Subnetz befinden.

Als letztes findet die Betrachtung der Werte des Beschleunigungssensors statt. Beim Spielen ist festzustellen, dass die Steuerung ausgesprochen genau reagiert. Wodurch ein geschicktes Ausweichen oder Einsammeln der Spielobjekte

[71] realisiert durch Airport Express und Airport Extreme

gegeben ist. Aus Abbildung 6.4 kann ein Ausschnitt aus dem Spiel mit den Beschleunigungswerten entlang der Y-Achse des iPhones betrachtet werden. Sehr schön zu sehen sind die gleichmäßigen Kurvenverläufe.

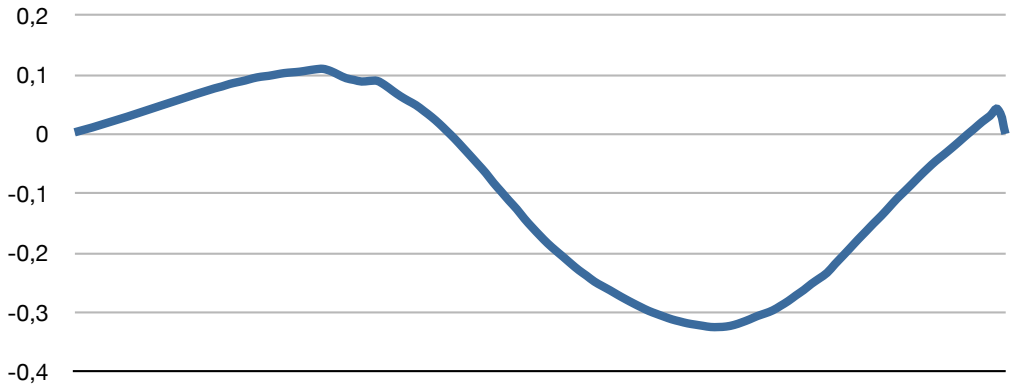

Abbildung 6.4.: Beschleunigungssensor mit Tiefpassfilter

Im Gegensatz dazu wurde in Abbildung 6.5 der Tiefpassfilter auf die Beschleunigungsdaten deaktiviert. Folglich sind jetzt extreme Sprüngen entlang der Kurve zu erkennen. Im Spielgeschehen äußert sich dies durch zittern der Spielfigur.

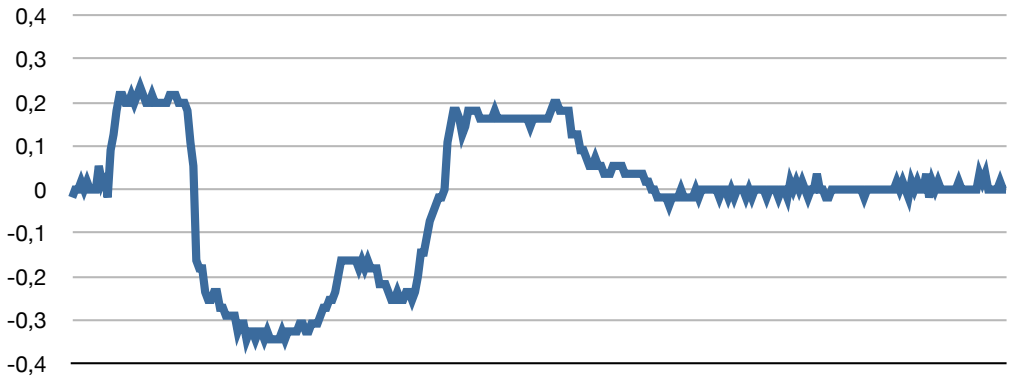

Abbildung 6.5.: Beschleunigungssensor ohne Tiefpassfilter

Abschließend kann erwähnt werden, dass die Darstellung der Inhalte mithilfe von OpenGL einen sehr sauberen Eindruck macht. Dies wurde auch erreicht

durch die Implementierung von Lichtquellen und Bestimmung der Materialeigenschaften der Spielobjekte und der Spielfigur. Für die Bedienung in den Menüs wurde auf die UI-Elemente der Cocoa API zurückgegriffen. Dadurch sind unter anderem sehr schöne Auswahlboxen möglich. Die Benutzung des Ganzen ist sehr intuitiv.

Durch das mitgelieferte Werkzeug *Instruments* können während der Entwicklungsphase bereits Objekt-Allokationen und Speicherleaks aufgedeckt werden. Dadurch kann die Speicherauslastung und Nutzung in der Anwendung erkannt werden. Da auf dem iPhone nur 64 MB an RAM zur Verfügung stehen, ist dies ein wichtiges Werkzeug bei der Entwicklung. Auch das Finden der Speicherleaks ist dadurch sehr einfach gestaltet. Für die Entwicklung auf dem iPhone steht leider kein *Garbage Collector* zur Verfügung, daher muss der Entwickler selber für das Freigeben von nicht mehr gebrauchten Speicher sorgen. Ferner ist auch die Überwachung der Prozessorauslastung ein sehr gelungenes Feature von Instruments.

6.2. Demonstration

Im folgenden Abschnitt werden vermehrt Screenshots eingesetzt um die Funktionalität des Spiels zu demonstrieren. Hierbei werden die verschiedenen Anwendungsfälle illustriert. Hier sei angemerkt, dass ein kleiner Darstellungsfehler in den Screenshots vorhanden ist. Die Funktionalität des 2 Spieler Modus setzt voraus das ein WLAN anliegt. Wenn ein Ad-hoc Netz durch ein MacBook erstellt wird, dann ändert das iPhone die Darstellung dessen nicht. Es bleibt bei der Anzeige einer 3G Verbindung obwohl ein WLAN anliegt.

6.2.1. 1 Player

Nach dem Start der Anwendung gelangt man zum Hauptmenü des Spiels. Hier hat man die Auswahl ein Solospiel oder 2-Player Spiel zu beginnen, außerdem wird die Möglichkeit offeriert sich den Highscore anzuschauen. Weiterhin wird hier die Option geboten die Tonausgabe mittels eines Sliders zu deaktivieren. Standardmäßig ist dieses auf *ON* gesetzt. Nach Betätigung des 1 Player Buttons kommt man direkt ins Spiel. Um nicht direkt mit entgegenkommenden Objekten konfrontiert zu werden, ist ein Countdown von 3 Sekunden vor dem Spielstart implementiert worden.

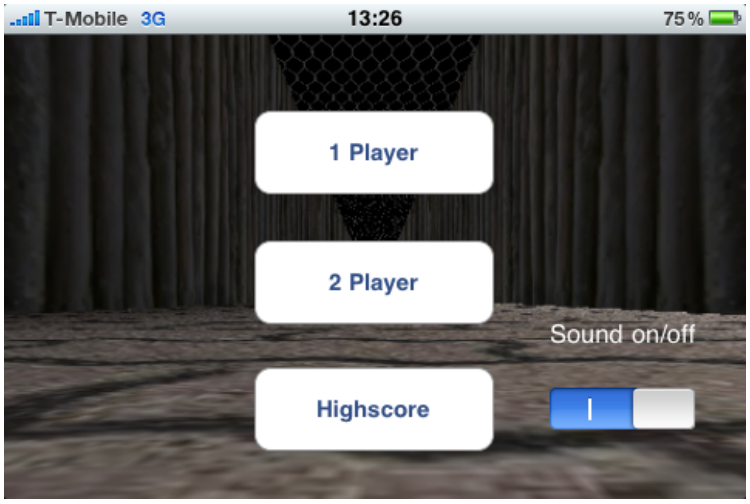

Abbildung 6.6.: Das Hauptmenü

Abbildung 6.7.: Der Countdown

Abbildung 6.8.: Das Spielgeschehen

Wenn der Spieler kein Leben mehr besitzt ist das Spiel zu Ende und die Datenbank wird nach der Platzierung abgefragt. Welche dann im Dialog ersichtlich ist. Außerdem hat der Nutzer die Wahl sich in die Datenbank einzutragen oder nicht. Abbildung 6.9 illustriert dies.

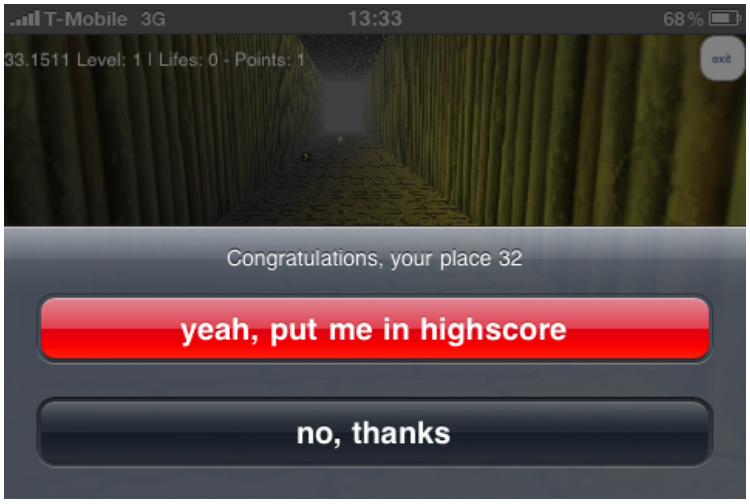

Abbildung 6.9.: Auswahlbox zum Eintrag in die Datenbank

Für den Fall, dass der Nutzer sich eintragen will, bekommt er ein Textfeld mit der Anweisung seinen Namen einzugeben, geboten. Zudem erscheint auch eine Meldung ob das Gerät die Positionsdaten ermitteln darf. Verneint er dieses wird der Highscore ohne Positionsangaben eingetragen.

Abbildung 6.10.: Ansicht zur Namenseingabe

Bei Berührung des Textfeldes wird dem Nutzer ein virtuelles Keyboard geboten, mit dem er seinen Namen eingeben kann. Beim drücken von *Fertig* wird die Eintragung vorgenommen.

Abbildung 6.11.: Ansicht zur Namenseingabe mit Tastatur

6.2.2. 2 Player

Das 2 Spielerspiel geht den gleichen Weg nur das im Vorfeld bestimmt werden muss, welche Rolle man einnehmen will. Für die Auswahl der Rolle (Client oder Host) wurden iPhone typische Auswahl Menüs implementiert.

Um dieses Menü sehen zu können, muss eine Verbindung zu einem WLAN anliegen. Andernfalls kommt eine Fehlermeldung, die auch entsprechend der Designrichtlinien von Apple implementiert worden ist. Diese Fehlermeldung (Vgl. Abbildung 6.13) gibt Aufschluss darüber, dass der Nutzer ein WLAN braucht um spielen zu können.

Durch die Auswahl ein Spiel zu hosten, erhält der Nutzer Feedback in Form eines Indikators (Vgl. Abbildung 6.14), der den Nutzer mitteilt, dass auf einen Gegenspieler gewartet wird. Sofern dieser gefunden wurde, startet das Spiel unmittelbar wieder mit einen Countdown wie aus Abbildung 6.7 ersichtlich ist.

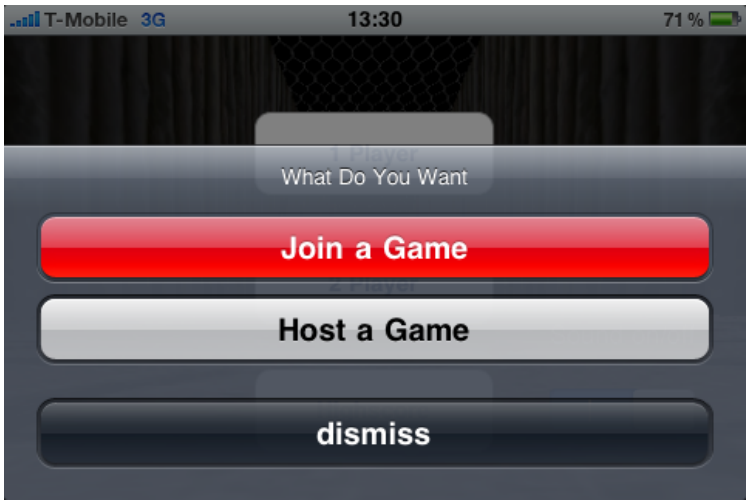

Abbildung 6.12.: Auswahlbox zum 2-Spieler Spiel

Sofern der Nutzer ein Spiel beitreten will, bekommt er eine Liste mit offenen Spielen zu sehen. Dort kann er sich für ein Spiel entscheiden und fängt unmittelbar nach der Auswahl an zu spielen. Abbildung 6.15 und 6.16 stellt dieses dar.

Nach dem Start des Spieles ist auch der zweite Spielball auf dem Spielfeld zu sehen. Erkennbar an dem schwarzen Ball aus Abbildung 6.17. Weiterhin ist gut zu erkennen wie die Kamera den Fokus auf die Spielfigur behält.

Für den Fall eines Verbindungsabbruches, wird wieder eine Fehlermitteilung dem Nutzer präsentiert (Vgl. Abbildung 6.18).

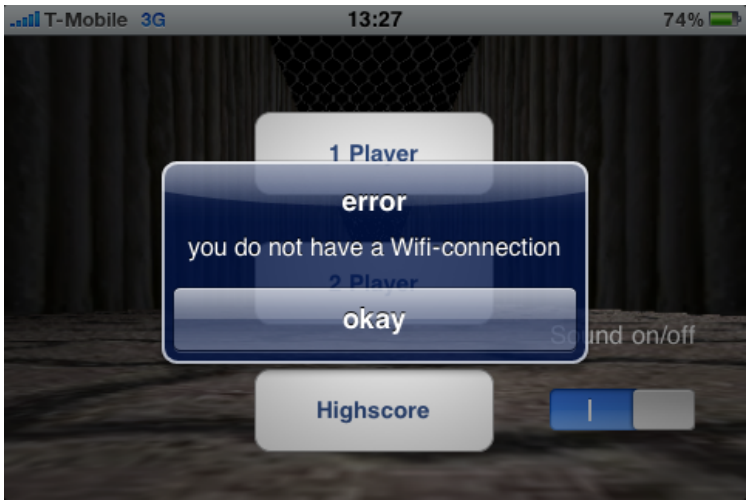

Abbildung 6.13.: Fehlermeldung

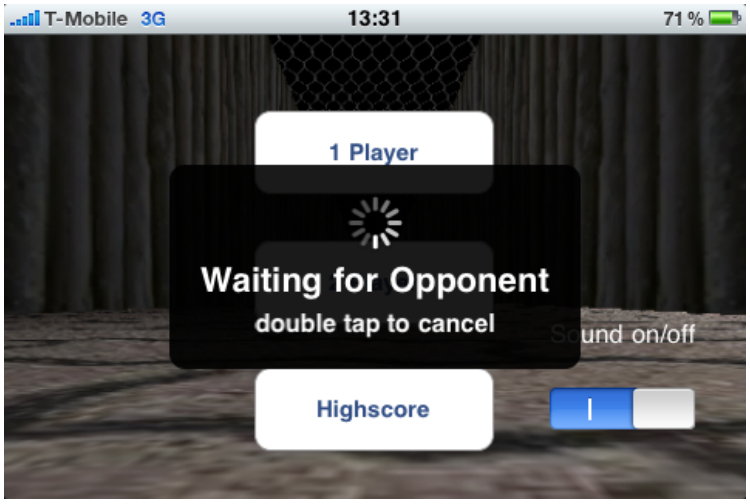

Abbildung 6.14.: Nutzerfeedback

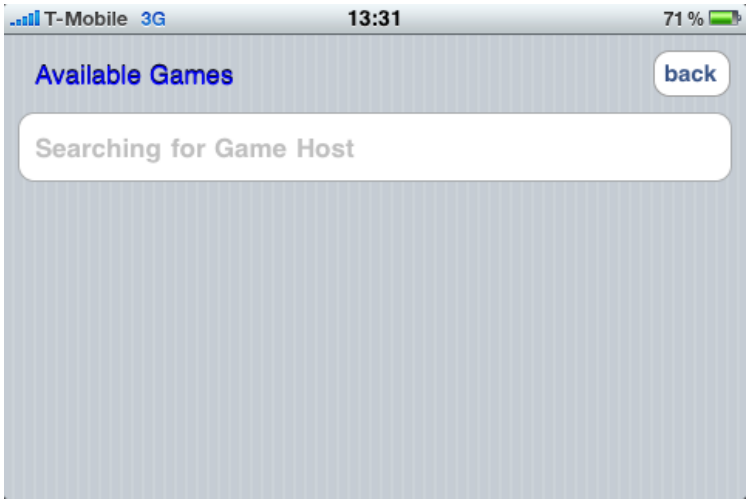

Abbildung 6.15.: kein Spiel zum Beitreten gefunden

Abbildung 6.16.: vorhandene offene Spiele

Abbildung 6.17.: Spielgeschehen mit 2ten Spieler

Abbildung 6.18.: Fehlermeldung bei verlorener Verbindung

6.2.3. Highscore

Der Higscore wird in Tabellenform dargestellt. Da je nach Verbindung eine gewisse Zeit vergehen kann bis der Highscore geladen ist, muss auch hier der Nutzer darüber informiert werden. Aus Abbildung 6.19 ist ersichtlich, dass mittels *retrieving Highscore* der Nutzer darüber unterrichtet wird.

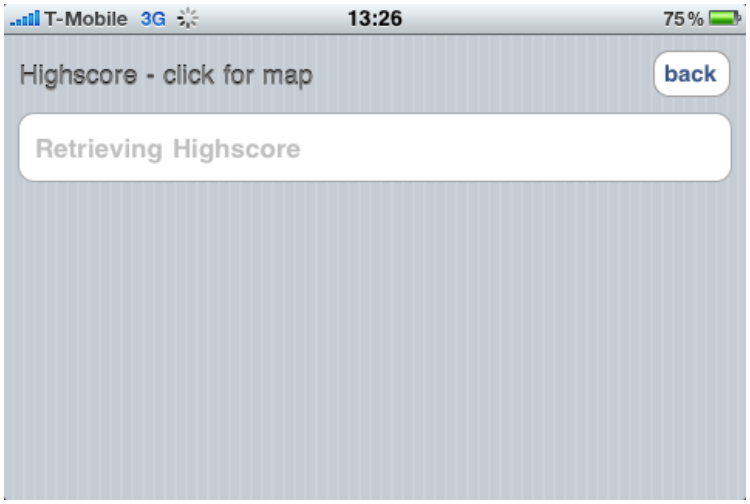

Abbildung 6.19.: nicht geladener Highscore

Sobald der Highscore geladen wurde, wird er dargestellt (Vgl. Abbildung 6.20).

Abbildung 6.20.: geladener Highscore

Bei Auswahl eines Eintrag öffnet sich anschließend das Programm Karten und zeigt die Position an dem der Highscore aufgestellt worden ist (Vgl. Abbildung 6.21).

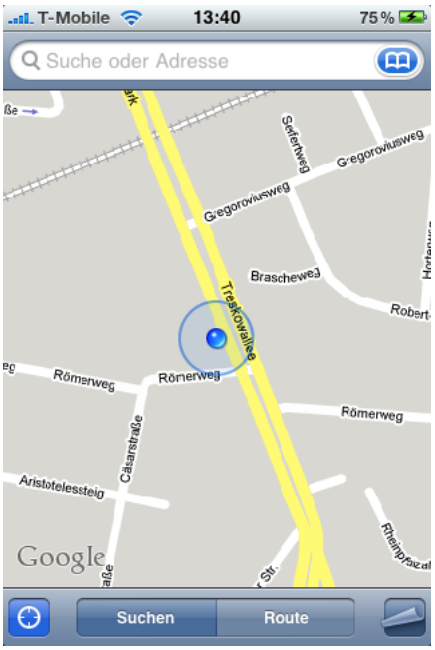

Abbildung 6.21.: Darstellung der Position innerhalb der Karten Anwendung

Ist für den Eintrag keine Position vorhanden, wird dies mittels Fehlermeldung dem Nutzer mitgeteilt (Vgl. Abbildung 6.22).

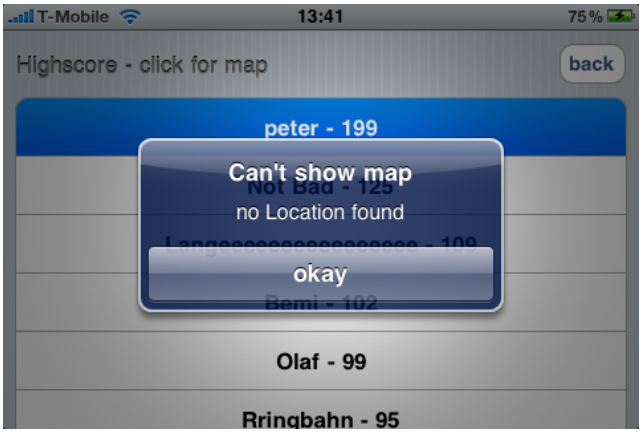

Abbildung 6.22.: Fehlermeldung über fehlende Positionsangaben

7. Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, ein Multiuser Spiel unter Nutzung verschiedener Sensordaten auf mobilen Geräten zu entwerfen und prototypisch umzusetzen.

Zu Beginn wurden zunächst die mobilen Geräte auf die Klasse der Smartphones eingeschränkt. Weiterhin wurde definiert, dass das Spielgeschehen innerhalb einer grafiklastigen 3D-Welt stattfinden soll. Demzufolge wurden anschließend die theoretischen Grundlagen zum Verständnis dieser Arbeit erfasst. Aus den Grundlagen heraus, ließ sich bereits ableiten, dass prinzipiell die Entwicklung auf mehreren Smartphones hätte stattfinden können, diese aber aufgrund der überwiegenden Vorteile des iPhones nur für dieses realisiert wurde.

Anschließend wurden im speziellen die Spiele auf Apple's App Store untersucht, um Tendenzen zu erkennen, die für den Systementwurf von Bedeutung waren. Aus den abgeleiteten Anforderungen wurde das Spiel konzipiert. Hier stand im Vordergrund eine ansprechende 3D-Spielwelt, sowie eine möglichst genaue Steuerung der Spielfigur zu realisieren. Hauptaugenmerk wurde auf die Netzwerkkommunikation gelegt, da sie in der Lage sein sollte die Realtime Anforderungen an das Spiel zu bewerkstelligen.

Der Systementwurf wurde nach dem modifizierten Cocoa MVC-Design Pattern entwickelt. So wurden im einzelnen die Datenhaltung, die Anwendungslogik, die Darstellung und die Kommunikation zwischen Client und Host entwickelt.

Die Umsetzung der 3D Spielwelt erfolgte mithilfe der OpenGL Bibliotheken. Der schwierigste Teil der Implementierung war die Realisierung der Netzwerkkommunikation. Diese stellte sich bei weitem aufwendiger heraus als gedacht. Anhand der Testergbnisse ließ sich aber feststellen, dass die Anforderungen an das Spiel erfüllt wurden.

Letztendlich kam ein Spiel zustande um ein kurzweiliges Spielvergnügen für Zwischendurch zu gewährleisten. Zudem bietet es ausreichend Spielspaß im 2 Spieler Betrieb. Das Ergebnis kann somit als Erfolg angesehen werden.

7.1. Ausblick

Ein Ausblick zur Weiterentwicklung des Spiels könnte die Implementierung einer verwaltenden Serverinstanz sein. Hier könnten sich Spieler registrieren und in einer Art Lobby sich Gegenspieler suchen. Dadurch würde die Mehrspielerfähigkeit übers das WLAN hinaus erweitert werden. Um hier eine Fairness im Spiel zu erreichen müssten Algorithmen zur Ausgleichung der unterschiedlichen Latenzen eingesetzt werden. Kontrollierende Instanz über den Gamestate würde dann der Server bilden. Weiterhin wäre durch die Spielerregistrierung eine eindeutige Zuordnung des Highscores zu einem Spieler gegeben.

Durch das Erscheinen des neuen Betriebssystems iPhone OS 3.0 während dieser Arbeit sind noch weitere Möglichkeiten für den Mehrspielermodus gegeben. Mit der neuen Version des Betriebssystem ist es möglich ein Ad-hoc Netzwerk mittels Bluetooth aufzubauen und auch den Datenaustausch darüber vorzunehmen. Das heißt es wäre keine Infrastruktur mehr in Form eines Routers notwendig um gegen seinen Freund anzutreten.

Durch den von Apple angebotenen *Push Notification Service* wäre auch das Erhalten einer Mitteilung, dass jemand gegen mich spielen will, auf dem iPhone denkbar. Dieser Service funktioniert auch selbst wenn die Applikation nicht gestartet ist. Durch die erhaltende Mitteilung kann ich das Spiel starten und gegen denjenigen, der die Einladung ausgesprochen hat, spielen.

Um den Spielpass zu erhöhen könnte die Physik eine größere Rolle im Spielablauf spielen. Kollisionen mit Wänden oder Spielgegner könnten in Verformungen oder Abstoßungen resultieren. Dies hätte zur Folge, dass der Algorithmus zur Kollisionsabfrage verbessert werden muss.

Literaturverzeichnis

Alle09 Allen, Mitch: *Palm webOS - Rough Cuts*. O'Reilly Media Inc., 2009. – S. 2–5. – ISBN 978–0–596–15525–4

ArClBr08 Armitage, Grenville ; Claypool, Mark ; Branch, Philip: *Networking and Online Games - Understanding and Engineering Multiplayer Internet Games*. Wiley Publishing, Inc., 2008. – S. 83–99. – ISBN 978–0–470–01857–6

BaLe Baughman, Nathaniel E. ; Levine, Brian N.: Cheat-Proof Playout for Centralized and Distributed Online Games / University of Massachusetts. – Forschungsbericht

Barr01 Barron, Todd: *Multiplayer Game Programming*. Prima Publishing, 2001. – S. 16–58. – ISBN 0–7615–3298–6

Berg04 Bergen, Gino van d.: *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, 2004. – S. 67–68. – ISBN 1–55860–801–X

BrSaBo06 Brun, J. ; Safaei, F. ; Boustead, P.: Managing Lantency and Fairness in Networked Games / University of Wollongong. 2006. – Forschungsbericht. – S. 2–13

Come01 Comer, Douglas E.: *Computernetzwerke und Internets mit Internet-Anwendungen*. 3. Pearson Education Deutschland GmbH, 2001. – S. 267 – 268. – ISBN 3–8273–7023–x

CoNo91 Cox, Brad J. ; Novobilski, Andrew J.: *Object-Oriented Programming - An Evolutionary Approach*. 2. Addison-Wesley Publishing Company, 1991. – S. VIII, 58. – ISBN 0–201–54834–8

Dalm03 DALMAU, Daniel Sanches-Crespo: *Core Techniques and Algorithms in Game Programming*. New Riders Publishing, 2003. – S. 231–251. – ISBN 0–1310–2009–9

Dunc03 DUNCAN, Andrew M.: *Objective-C Pocket Reference - A Guide to Language Fundamentals*. Bd. 1. O'Reilly and Associates, Inc., 2003. – ISBN 0–596–00423–0

Eric05 ERICSON, Christer: *Real-Time Collision Detection*. Morgan Kaufmann Publishers, 2005. – S. 214–228. – ISBN 1–55860–732–3

Jung07 JUNG, Thomas: *Vorlesungsfolien*. Thomas Jung, 2007. – Transformation, Verdeckung, Beleuchtung, Texturierung S. – ISBN keine

LaMa09 LAMARCHE, J. ; MARK, D.: *The Beginning of iPhone Development*. Apress, 2009. – S. 361–362. – ISBN 978–1–4302–1626–1

Lipi98 LIPINSKI, Klaus: *Lexikon Lokale Netze*. Internation Thomson Publ., 1998. – S. 43, 107. – ISBN 3–8266–4037–3

Lowe05 LOWE, Doug: *Networking - For Dummies*. 2. Wiley Publishing, Inc., 2005. – S. 15–17. – ISBN 0–7645–9939–9

Mauv00 MAUVE, Martin: How to Keep a Dead Man from Shooting / University of Mannheim. 2000. – Forschungsbericht

McMa06 MCMAHON, David: *Linear Algebra Demystified*. The McGraw-Hill Companies, 2006. – ISBN 0–07–146579–0

PeDa07 PETERSON, Larry L. ; DAVIE, Bruce S.: *Computernetzwerke - Eine systemorientierte Einführung*. Bd. 4. dpunkt.verlag, 2007. – S. 714 – 715; 384–395; 133–148. – ISBN 978–3–89864–491–4

PuAaMi+08 PULLI, Kari ; AARNIO, Tomi ; MIETTINEN, Ville ; ROIMELA, Kimmo ; VAARALA, Jani: *Mobile 3D Graphics with OpenGL ES and M3G*. Morgan Kaufmann Publishers, 2008. – S. 18–79. – ISBN 978–0–12–373727–4

RO03 R. Osswald, Prof. Dr.-Ing. habil.: *Vorlesungsfolien*. Prof. Dr.-Ing. habil. R. Oßwald, 2003. – Datenbank / Informationssysteme S. – ISBN keine

ScLi07 Schemberg, A. ; Linten, M.: *PC-Netzwerke*. Bd. 4. Galileo Press, 2007. – S. 27–31; 50–66. – ISBN 978–3–8362–1062–1

Stei08 Stein, Erich: *Rechnernetze und Internet*. Bd. 3. Fachbuchverlag Leipzig im Carl Hanser Verlag, 2008. – 39, 137–141 S. – ISBN 978–3–446–40976–7

Tane04 Tanenbaum, Andrew S.: *Computernetzwerke*. 4. Pearson Education Deutschland GmbH, 2004. – S. 18, 31–67; 86–90. – ISBN 3–8273–7046–9

Internetquellen

1 148Apps.biz: *App Store Metrics*. `http://148apps.biz/app-store-metrics/`. Version: 2009, Abruf: 08.07.2009

2 Alliance, Wi-Fi: *Get to Know the Alliance*. `http://www.wi-fi.org/about_overview.php`. Version: 2008, Abruf: 23.06.2009

3 Apple Computer, Inc.: *Object-Oriented Programming with Objective-C*. `http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/OOP_ObjC/Articles/ooWhy.html`, Abruf: 13.06.2009

4 Apple Computer, Inc.: *Bonjour Overview*. `http://developer.apple.com/documentation/Cocoa/Conceptual/NetServices/NetServices.pdf`. Version: Mai 2006, Abruf: 14.06.2009

5 Apple Computer, Inc.: *Apple Human Interface Guidelines*. `http://developer.apple.com/documentation/userexperience/Conceptual/AppleHIGuidelines/XHIGIntro/XHIGIntro.html`. Version: 2007, Abruf: 07.07.2009

6 Apple Computer, Inc.: *The Objective-C Programming Language*. `http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf`. Version: 2007, Abruf: 11.06.2009

7 Apple Computer, Inc.: *Cocoa Fundamentals Guide*. `http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html`. Version: 2009, Abruf: 28.06.2009

8 Apple Computer, Inc.: *Technical Specifications*. `http://www.apple.com/iphone/specs.html`. Version: 2009, Abruf: 20.06.2009

9 DPA: *Palm setzt mit Pre alles auf eine Karte*. `http://www.n24.de/news/newsitem_5113116.html`. Version: 2009, Abruf: 19.06.2009

10 GLBenchmark: *Smartphones Performance Details.* http://www.glbenchmark.com/phonedetails.jsp, Abruf: 19.06.2009

11 Holtkamp, Heiko: *TCP/IP im Detail.* https://ernstweiler.de/docs/tcp-ip/tcp-ip3/kap_2_4.html. Version: 2003, Abruf: 18.06.2009

12 kowoma.de: *GPS-Anwendung.* http://www.kowoma.de/gps/. Version: 2009, Abruf: 20.06.2009

13 Kranz, Matthias: *Sensoren, Microcontroller und Ubiquitous Computing.* http://www.hcilab.org/matthias/SensorenMicrocontrollerUbiquitousComputing.pdf. Version: 2004, Abruf: 19.06.2009

14 Kremp, Matthias: *Der Spiegel: WPS statt GPS.* http://www.spiegel.de/netzwelt/tech/0,1518,528797-2,00.html. Version: 2008, Abruf: 20.06.2009

15 Kremp, Matthias: *Der Spiegel: Sommer der Smartphones.* http://www.spiegel.de/netzwelt/mobil/0,1518,629420,00.html. Version: 2009, Abruf: 07.07.2009

16 Lount, Peter W.: *A Brief Introduction to Smalltalk.* http://www.smalltalk.org/articles/article_20040000_11.html, Abruf: 11.06.2009

17 MacPrime: *Die NeXT-Story.* http://www.macprime.ch/applehistory/story/die-next-story/P3/, Abruf: 13.06.2009

18 Paun, Christopher: *Streit über UMTS-Vergabepraxis.* http://www.teltarif.de/arch/2000/kw18/s2059.html. Version: 2000, Abruf: 23.06.2009

19 Pittenauer, Martin ; Wagner, Dominik: *Podcast: iPhone Anwendlungsentwicklung.* http://chaosradio.ccc.de/cre101.html. Version: 2008, Abruf: 14.06.2009

20 Plate, Prof. J.: *Grundlagen Computernetze.* http://www.netzmafia.de/skripten/netze/netz0.html. Version: 2007, Abruf: 16.06.2009

21 SCHULTZ, Stefan: *Der Spiegel: Revolution liegt in der Luft*. `http://www.spiegel.de/wirtschaft/0,1518,607872,00.html`. Version: 2009, Abruf: 07.07.2009

22 STMICROELECTRONICS: *MEMS motion sensor 3-axis - ± 2g/± 8g smart digital output "piccolo" accelerometer*. `http://www.st.com/stonline/products/literature/ds/12726.pdf`. Version: 2008, Abruf: 20.06.2009

23 WIKIPEDIA: *Wikipedia Eintrag zu Client-Server-Modell*. `http://de.wikipedia.org/wiki/Client-Server-Modell`, Abruf: 16.06.2009

24 WIKIPEDIA: *Wikipedia Eintrag zu Objective-C*. `http://en.wikipedia.org/wiki/Objective-C`, Abruf: 11.06.2009

25 WIKIPEDIA: *Wikipedia Eintrag zu Peer-to-Peer-Modell*. `http://de.wikipedia.org/wiki/Peer-to-Peer`, Abruf: 16.06.2009

26 WILSON, Tracy V.: *How the iPhone Works*. `http://electronics.howstuffworks.com/iphone.htm`, Abruf: 19.06.2009

27 WIRELESS, Skyhook: *Hybrid Positioning System (XPS)*. `http://skyhookwireless.com/howitworks/xps.php`, Abruf: 20.06.2009

A. Anhang

Listing A.1: die gluLookAt Funktion

```
void gluLookAt(GLfloat eyex, GLfloat eyey, GLfloat eyez, GLfloat centerx, GLfloat centery,
     GLfloat centerz, GLfloat upx, GLfloat upy, GLfloat upz){
    GLfloat m[16];
    GLfloat x[3], y[3], z[3];
    GLfloat mag;
    /* Make rotation matrix */
    /* Z vector */
    z[0] = eyex - centerx;  z[1] = eyey - centery;  z[2] = eyez - centerz;
    mag = (float)sqrt(z[0] * z[0] + z[1] * z[1] + z[2] * z[2]);
    if (mag) {
        z[0] /= mag;        z[1] /= mag;       z[2] /= mag;
    }
    /* Y vector */
    y[0] = upx;   y[1] = upy;   y[2] = upz;
    /* X vector = Y cross Z */
    x[0] = y[1] * z[2] - y[2] * z[1];
    x[1] = -y[0] * z[2] + y[2] * z[0];
    x[2] = y[0] * z[1] - y[1] * z[0];
    /* Recompute Y = Z cross X */
    y[0] = z[1] * x[2] - z[2] * x[1];
    y[1] = -z[0] * x[2] + z[2] * x[0];
    y[2] = z[0] * x[1] - z[1] * x[0];
    mag = (float)sqrt(x[0] * x[0] + x[1] * x[1] + x[2] * x[2]);
    if (mag) {
        x[0] /= mag;       x[1] /= mag;       x[2] /= mag;
    }
    mag = (float)sqrt(y[0] * y[0] + y[1] * y[1] + y[2] * y[2]);
    if (mag) {
        y[0] /= mag;       y[1] /= mag;       y[2] /= mag;
    }
#define M(row,col) m[col*4+row]
    M(0, 0) = x[0];   M(0, 1) = x[1];    M(0, 2) = x[2];
    M(0, 3) = 0.0;    M(1, 0) = y[0];    M(1, 1) = y[1];
    M(1, 2) = y[2];   M(1, 3) = 0.0;     M(2, 0) = z[0];
    M(2, 1) = z[1];   M(2, 2) = z[2];    M(2, 3) = 0.0;
    M(3, 0) = 0.0;   M(3, 1) = 0.0;      M(3, 2) = 0.0;
    M(3, 3) = 1.0;
#undef M   {
        int a;
        GLfloat fixedM[16];
        for (a = 0; a < 16; ++a)
            fixedM[a] = m[a];
        glMultMatrixf(fixedM);
    }
    /* Translate Eye to Origin */
    glTranslatef(-eyex, -eyey, -eyez);
}
```

Listing A.2: die gluPerspective Funktion

```
1 void gluPerspective(GLfloat fovy, GLfloat aspect, GLfloat zNear, GLfloat zFar){
2   GLfloat xmin, xmax, ymin, ymax;
3   ymax = zNear * (GLfloat)tan(fovy * M_PI / 360);
4   ymin = -ymax;
5   xmin = ymin * aspect;  xmax = ymax * aspect;
6   glFrustumf(xmin, xmax, ymin, ymax, zNear, zFar);
7 }
```

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Index

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

> *Konzeption und prototypische Implementierung eines Multiuser Spiels unter Nutzung von Sensordaten auf mobilen Geräten*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den 23. Juli 2009

Carsten Guhl