Institut für parallele und verteilte Systeme

Abteilung für verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2427

Entwicklung der OpenGL-basierten grafischen Oberfläche eines E-learning-Spiels mit Fokus Regelungstechnik

Dominik Herr

Studiengang:	Dipl. Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Betreuer:	IST: C. Breindl, A. Kramer, P. Weber
begonnen am:	1.12.2012
beendet am:	25.4.2013
CR-Klassifikation:	H.5.2, I.2.6, I.3.3, I.3.5, I.3.6, J.2

Kurzfassung

In dieser Arbeit geht es um den konzeptionellen Entwurf und die Implementierung eines auf dem Betriebssystem Android basierten Spiels mit dem Fokus auf den Entwurf eines Reglers. Dieses Thema wurde in zwei Studienarbeiten aufgeteilt. In dieser Arbeit liegt der Fokus auf der Konzipierung und der Implementierung der visuellen Elemente des Kernspiels, sowie des Entwurfs von Highscore und Spielzusammenfassung. Des Weiteren wird auf die Schnittstellen zwischen der Visualisierungskomponente dieser Studienarbeit und der zweiten Studienarbeit, in welcher der Regler entworfen und implementiert wurde, eingegangen.

Diese Arbeit stellt nach einer kurzen Einleitung und den notwendigen Grundlagen der Regelungstechnik und des Betriebssystems Android zunächst im Konzept die Architektur des Spiels und die dabei benötigten Komponenten vor. Hierbei wird auch auf die genutzten Konzepte und Entwurfsmuster der Softwaretechnik eingegangen. Im anschließenden Implementierungskapitel wird zunächst der Ablauf des Spiels skizziert. Dann wird vor allem auf die konkrete Implementierung der Visualisierungskomponente im Detail eingegangen. Im Anschluss werden die notwendigen Schnittstellen zwischen Regler- und Visualisierungskomponente, sowie deren konzeptionelle Probleme und Lösungsvorschläge vorgestellt. Zuletzt wird auf die Verwaltung und Visualisierung des Highscores eingegangen.

Abstract

The goal of this work is the conceptual design and the implementation of an android-based game with a focus on the design of a system controller. This goal is split into two parts. This work's focus is the concept and implementation of the visual elements of the core game as well as the design of the highscore and the game's summary. Additionally, the necessary interface between the visualization and the system controller, which is the focus of the second student thesis, will be shown.

This student thesis first presents a short introduction and the necessary basics of system controllers and the operation system Android. Afterwards the concept chapter deals with the architecture of the game and its components. In this context the used concepts and design patterns used in software engineering will be presented. The following chapter shows the conceptual course of the game. After that the Implementation of the visualization component will be presented in detail. Subsequently the interface between the system controller and the visualization as well as its conceptual problems and solutions will be presented. At last the management and visualization of the highscore will be shown.

Inhaltsverzeichnis

1. Einleitung **3**

2. Grundlagen **5**
2.1. Grundlagen der Regelungstechnik 5
2.2. Grundlagen der Architektur des Betriebssystems Android 6
2.2.1. Was ist Android? 6
2.2.2. SDK, Eclipse und ADT 6
2.2.3. Einrichten von Computer und Smartphone 7
2.2.4. Gerätetreiber 9
2.2.5. Activity Lifecycle 9
2.2.6. Einführung in OpenGL ES unter Android 12

3. Konzept **17**
3.1. Architektur des Spiels 17
3.1.1. Aufbau 17
3.1.2. Genutzte Konzepte und Design-Patterns 18
3.1.3. Controller 21
3.1.4. Sensor 21
3.1.5. Menü 22
3.1.6. Grafische Oberfläche 23
3.1.7. Zusammenfassungsbildschirm 25
3.1.8. Highscore 25

4. Implementierung **27**
4.1. Spielphasen 27
4.1.1. Auswahl des Spielmodus 28
4.1.2. Festlegung der Parameter des Reglers 28
4.1.3. Optional: Anzeige von Pol- und Bodediagramm 30
4.1.4. Spiel 30
4.1.5. Spielzusammenfassung 30
4.1.6. Highscore 31
4.2. Implementierung der grafischen Darstellung des Modells 31
4.2.1. Zeitliche Diskretisierung der Aktualisierung der Daten des grafischen Elemente 32
4.2.2. Sterne 32
4.2.3. Hintergrund 33
4.2.4. Dynamische Erstellung neuer grafischer Elemente 33
4.2.5. Darstellung der Vorgabefunktion 34

4.2.6. Visualisierung des Verlaufs des Raumschiffs . . . 34
4.2.7. Raumschiff . . . 34
4.2.8. Leistungsoptimierung . . . 36
4.3. Schnittstelle zwischen Modell und Visualisierung . . . 37
4.3.1. Schnittstellen zwischen Modell und Visualisierung . . . 37
4.3.2. Konzeptionelle Probleme der Schnittstelle . . . 37
4.3.3. Mögliche Lösungsansätze und deren Vor- und Nachteile . . . 38
4.4. Highscore . . . 38
4.4.1. Verwaltung des Highscores . . . 39
4.4.2. Visualisierung des Highscores . . . 39

5. Zusammenfassung und weiterer Ausblick **41**

A. Grundlagen der Regelungstechnik **43**
A.1. Modellgleichung / Modellklasse . . . 43
A.2. Pol-Diagramm . . . 44
A.3. Nyquistdiagramm . . . 44
A.3.1. Satz (Nyquist-Stabilitatskriterium) . . . 45
A.4. Bodediagramm . . . 45

B. Systemarchitektur des Spiels **47**

Literaturverzeichnis **49**

1. Einleitung

Spielerisches Lernen ist ein effizientes und sinnvolles Mittel, um Lehrinhalte an junge Menschen zu vermitteln [18]. Neuere Studien schlagen vor, man könne solche Mittel in die aktive Lehre einbinden und dadurch das Lehrangebot erweitern. Die Umsetzung dieser Idee scheitert häufig vor allem an einer ausreichenden Verteilung von elektronischen Geräten, die ein solches Angebot unterstützen würden. In letzter Zeit ergibt sich jedoch durch die immer weitere Verbreitung von Smartphones [9] die Möglichkeit, eine verhältnismäßig große Gruppe von Menschen durch die Verteilung einer App zu erreichen.
Mit diesem Hintergrund hat sich das Institut für Systemtheorie und Regelungstechnik (IST) entschieden, ihr bereits existierendes Angebot an Regelungstechnikspielen, die allesamt browserbasiert waren, dahingehend zu erweitern, dass man den wachsenden Smartphonemarkt [11] nutzen möchte. Da das freie Smartphone-Betriebssystem Android in Deutschland zwischenzeitlich das am weitesten verbreitete System auf dem Markt ist [2] lag es nahe, für dieses System zu entwickeln. Da theoretisch 97,6% aller Android-basierten Systeme Apps unterstützen, die für die Version 2.2 von Android programmiert wurden, ist das Spiel, das in dieser Studienarbeit behandelt wird, mit dieser Android Distribution kompatibel.

Das Ziel dieser Studienarbeit in Verbindung mit der Studienarbeit von Michael Merg ist es, ein android-basiertes Regelungstechnikspiel zu entwickeln. Dieses soll für Studenten im Gebiet der Regelungstechnik einen spielerischen Lernaspekt darstellen und bei Studenten ohne oder mit wenig Erfahrung im Gebiet der Regelungstechnik Interesse fördern beziehungsweise das Verständnis erhöhen.

Zudem finden sich im Internet praktisch keine wissenschaftlich relevanten Arbeiten, die das Thema der Regelungstechnik mit Smartphones verbindet. Die meisten Firmen nutzen Smartphones nur, um Informationen anzuzeigen (beispielsweise den Energieverbrauch einer Heizung). Die Verbesserung der User Experience mit Hilfe von Smartphones wird in diesem Themengebiet weitestgehend ignoriert.

Das App wurde thematisch in zwei Gebiete unterteilt, woraus sich zwei Studienarbeiten ergaben, die zusammen betrachtet das vollständige Spiel beschreiben. Diese Arbeit befasst sich hauptsächlich mit den Grundlagen des Betriebssystems Android, der Visualisierung des eigentlichen Spiels und derer effizienten Verwaltung. Des Weiteren wird in dieser Studienarbeit auf die Entwicklung und Verwaltung des Highscore-Mechanismus des Spiels eingegangen. In der Arbeit von Michael Merg [13] liegt der Fokus auf den regelungstechnischen Hintergründen des Spiels und auf der Menüführung. Außerdem wird dort auf die zusätzlichen Visualisierungen vor dem Spielstart eingegangen. Die regelungstechnischen Grundlagen werden zusätzlich in Anhang A beschrieben. In beiden Arbeiten wird auf die Schnittstellen der beiden Arbeiten und deren Vor- und Nachteile eingegangen.

2. Grundlagen

Da die App auf zwei Arbeiten basiert gibt es auch zwei Grundlagenkapitel. Da die Grundlagen der Regelungstechnik nur für das Gesamtkonzept benötigt werden und diese Arbeit nicht direkt davon betroffen ist, sind die Grundlagen der Regelungstechnik als Kurzfassung der Arbeit von Michael Merg im Anhang zusammengefasst.

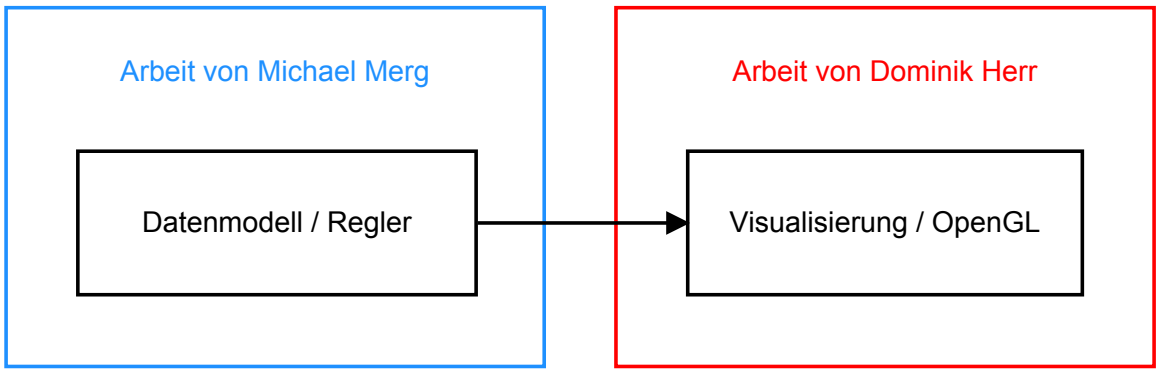


Abbildung 2.1.: Übersicht über die Aufteilung der Grundlagen.

2.1. Grundlagen der Regelungstechnik

Die Grundlagen der Regelungstechnik sind nicht Gegenstand dieser Studienarbeit. Sie werden zwar in dieser Studienarbeit nicht direkt benötigt oder verwendet, sie sind aber dennoch wichtig für das Gesamtverständnis der App, vor allem von deren Datenmodell. Aus diesem Grund können die wichtigsten Punkte wie Modell, Modellgleichungen, Zustandsrückführung, Beobachter, sowie die Pol-, Nyquist- und Bodediagramme in Anhang A nachgeschlagen werden. Der Hauptteil des Grundlagenkapitels der Regelungstechnik entstammt der Arbeit von Michael Merg [13] und wird dort ausführlicher beschrieben.

2.2. Grundlagen der Architektur des Betriebssystems Android

Das folgende Kapitel soll eine Übersicht über das Betriebssystem Android geben und dem Leser die Einrichtung und Konfiguration einer Entwicklungsumgebung ermöglichen, mit der man Apps für Android entwickeln und installieren kann.

2.2.1. Was ist Android?

„Android ist eine Ansammlung von Software, die ein Betriebssystem sowie Middleware und Schlüsselprogramme enthält.“ [8]
Android ist ein von Google Inc. entwickeltes Betriebssystem, dass speziell für mobile Endgeräte wie Smartphones und Tablets entwickelt wurde. Es ist dabei bereits mit einer gewissen Grundfunktionalität ausgestattet. Zu dieser gehört jedoch explizit nicht die eigentliche Oberfläche. Aufgrund dieses Umstands ist es möglich, dass gewisse Funktionen bei unterschiedlichen Smartphones an unterschiedlichen Stellen aufzufinden sind. Dies ist zwar wichtiges Hintergrundwissen, für diese Arbeit jedoch nur von geringer Bedeutung, da es nur die eigentliche Installation des Programmes betreffen könnte. Android zeichnet sich insbesondere dadurch aus, dass es quelloffen ist. Das bedeutet, dass jeder Mensch, der die Programmiersprache Java beherrscht, den Programmcode des Android-OS einsehen und unabhängig weiterentwickeln kann. Für einen Anwender mögen diese Informationen bereits genügen, doch eigentlich ist der Zweck dieser Arbeit, sowohl ein Grundverständnis von Abläufen des Betriebssystems, als auch die Wartung und Weiterentwicklung des SpaceController-Apps zu ermöglichen. Dazu muss man allerdings zunächst eine Entwicklungsumgebung einrichten.

2.2.2. SDK[1], Eclipse und ADT[2]

Google hat für sein Betriebssystem Werkzeuge veröffentlicht, die es dem Benutzer ermöglichen sollen, Programme für Android zu entwickeln. Die drei wichtigsten werden im Folgenden kurz vorgestellt.

1. Android SDK: Ein Software Development Kit (SDK) ist eine Sammlung von Werkzeugen und Anwendungen, die dazu dienen, eine Software zu erstellen, meist inklusive Dokumentation [20]. Im Falle von Android gibt es für fast jede Version des Betriebssystems ein eigenes SDK, wobei in der Regel eine höhere Version des Betriebssystems mit mehr Funktionalität einher geht. Es kommt jedoch auch vor, dass Funktionen aus älteren Versionen von Android als veraltet eingestuft werden und nicht mehr unterstützt werden. Aufgrund dieser Tatsache sei darauf hingewiesen, dass die SpaceController App für Android 2.3.3 (entspricht API[3]-Level 10) entwickelt wurde. Auf Smartphones mit Android 4.x scheint das Spiel problemlos zu laufen (getestet mit einem Samsung Galaxy Nexus und einem Samsung Galaxy S3 LTE). Es wurde nicht unter Android 1.x und 3.x getestet (Android 3 entspricht der Tablet-Version von

[1] SDK: Software Development Kit
[2] ADT: Android Developer Tools
[3] API: Application Programming Interface (engl.) ≙ Programmierschnittstelle (dt.)

Android, die jedoch kaum Verbreitung hat, da die meisten Tablets heutzutage mit Android 4.x laufen). Unter Android 2.1.x wurde es mit einem Smartphone getestet und es funktionierte, wenn auch nur mit eingeschränkter Leistung.

2. Eclipse IDE[4]: In diesem Zusammenhang vereinfacht gesagt, ist das Eclipse IDE eine Entwicklungsumgebung, welche die Programmiersprache Java gut unterstützt. Android-Apps werden in der Regel mit der Programmiersprache Java geschrieben, weshalb sich die Nutzung von Eclipse als Entwicklungsumgebung anbietet. Es gibt keinen Zwang dies zu tun, aber Google hat für Eclipse ein Plugin zur Verfügung gestellt, mit dem man mit Eclipse Android-Apps schreiben und viele der mitgelieferten Tools bequem erreichen und verwenden kann. Alternativ ist es auch möglich, Android in anderen Programmiersprachen zu programmieren. In diesem Fall ist es möglich, dass sich andere Entwicklungsumgebungen besser eignen als Eclipse. So ist es beispielsweise möglich, Android-Apps mit C# (in Form von *Mono* [14]) zu programmieren und es existiert ein „Native Development Kit (NDK)“, mit dem man in Apps C und C++ Bibliotheken einbinden kann. Aufgrund des zusätzlichen Aufwands findet man jedoch auf der Seite des NDK den Hinweis “the NDK will not benefit most apps“ [5].

3. Android ADT: Die Android Developer Tools (kurz ADT) sind ein Plugin für das Eclipse IDE. Es liefert dabei direkten Zugriff auf viele der vom Android SDK gelieferten Tools sowie grafische Benutzeroberflächen (kurz GUI) für manche der Kommandozeilen-basierten Programme. Außerdem bietet es viele Hilfen für die Entwicklung des Android-Apps an sich. So ist im ADT ein Assistent enthalten, der dem Benutzer ein neues Projekt anlegt und es gibt einen GUI-Designer. Des Weiteren können mit Hilfe des ADT die SDKs von Android in Eclipse eingebunden werden, wodurch eine auto-complete Funktionalität zur Verfügung gestellt wird. Zudem kann man hierdurch Android-Apps direkt von Eclipse aus auf ein simuliertes Gerät oder ein via USB angeschlossenes Smartphone übertragen.

2.2.3. Einrichten von Computer und Smartphone

Die im vorherigen Abschnitt vorgestellten Tools müssen jetzt noch heruntergeladen, installiert und eingerichtet werden. *Vorbereitung:* Als Grundvoraussetzung benötigt man das JDK[5], welches unter [6] heruntergeladen werden kann.

Nachdem das JDK installiert wurde, lädt man unter [7] zunächst das jeweilige SDK herunter, dass zu dem Betriebssystem passt, mit dem entwickelt werden soll. Im Falle von Windows kann man das Installationsprogramm herunterladen und das SDK an den Ort der Wahl installieren. Man sollte dabei darauf achten, dass der Installationsort des SDK bekannt ist (unter anderem, da der Standardpfad zwar sinnvoll, aber nicht trivial ist). Diese Kurzeinführung geht im Folgenden davon aus, dass Eclipse IDE als Entwicklungsumgebung verwendet wird. Falls man eine andere Entwicklungsumgebung

[4] IDE: Integrated Development Environment (engl.) ≙ Integrierte Entwicklungsumgebung (dt.)
[5] JDK: Java Development Kit
[6] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[7] http://developer.android.com/sdk/index.html

nutzen möchte, finden sich bei Android Developers [6] entsprechende Anleitungen. Um Eclipse zu verwenden, lädt man am Besten unter `http://eclipse.org/downloads/` die *classic*-Version von Eclipse herunter. Das Archiv extrahiert man an einen beliebigen Ort und startet Eclipse wie jedes andere Programm. In Eclipse lädt man nun mittels des Update-Managers das ADT-Plugin herunter. Den Manager erreicht man unter [Help] → [Install New Software...]. Die Internet-Adresse des ADT ist `https://dl-ssl.google.com/android/eclipse/`.

Nachdem ADT installiert und Eclipse neu gestartet wurde, geht man in die Android-Einstellungen (zu finden unter Window → Preferences → Android). Dort muss man den Pfad des am Anfang heruntergeladenen Android-SDK angeben. Daraufhin kann man in Eclipse den Android SDK Manager aufrufen (siehe Abb. 2.2). Dort muss man nun die Android-Umgebungen herunterladen, welche man nutzen will. Anmerkung: Eclipse

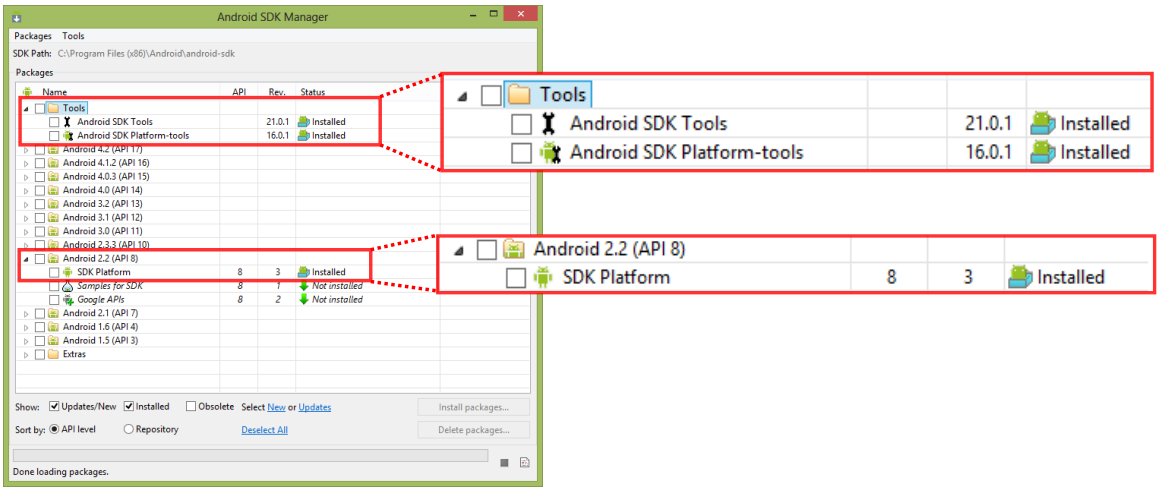


Abbildung 2.2.: Android SDK Manager mit Markierungen bei notwendigen Paketen.

bietet außerdem Zugriff auf den Android-Emulator AVD[8], auf den hier jedoch nicht näher eingegangen wird, da dessen Performanz zu schlecht ist, um ein OpenGL-App wie das Android Spiel dieser Arbeit zu testen, zu debuggen oder zu spielen.

Jetzt, da der Computer für die Android-Entwicklung eingerichtet ist, muss man noch auf dem Smartphone den Entwickler-Modus aktivieren. Dazu ruft man die Einstellungen auf, navigiert zum Eintrag *Entwickler* oder *Entwicklung* (sowohl Name als auch Ort sind geräteabhängig, jedoch findet man den Entwicklereintrag entweder direkt in den Einstellungen oder unter dem Eintrag *Anwendungen*). Dort muss man die Checkbox *USB-Debugging* aktivieren. Dies ermöglicht es, Nicht-Market-Apps auf dem Smartphone zu installieren. Es besteht seit Android 4.x die Möglichkeit, dass der Entwicklermodus überhaupt nicht für einfache Benutzer sichtbar ist. In diesem Fall sollte man im Internet nachschlagen, wie man mit seinem Smartphone den Entwicklermodus wieder freischaltet.

[8] AVD: Android Virtual Device

2.2.4. Gerätetreiber

Damit ein Android-basiertes Smartphone von dem jeweiligen Betriebssystem erkannt wird, muss man (je nach Gerät) möglicherweise einen Treiber nachinstallieren. Diesen hat man entweder bereits auf der Festplatte (wenn er mit dem Android-SDK heruntergeladen wurde, dann befindet er sich im SDK-Verzeichnis im Unterordner ./google/usb_driver), oder man muss ihn aus dem Internet herunterladen. Die besten Internet-Quellen für den benötigten Treiber im Internet sind Android Developers [7] und der Hersteller des Smartphones. In manchen Fällen ist es nicht möglich, den Treiber direkt herunterzuladen. In diesen Fällen bietet der Smartphonehersteller meist ein Verwaltungsprogramm an, welches die Treiber enthält.

2.2.5. Activity Lifecycle

Im Folgenden wird erläutert, was man unter einer Activity in Android versteht, wie Activities verwaltet werden und wie man eine Activity aufruft. Anschließend wird darauf eingegangen, wie der Lebenszyklus einer Activity aufgebaut ist und wofür die verschiedenen für die App relevanten Phasen gedacht sind. Zuletzt wird erklärt, wie man Informationen von einer momentan sichtbaren Activity an die Activity weitergeben kann, von der sie aufgerufen wurde. Zur besseren Übersicht ist der Activity Lifecycle in Abbildung [2.3] dargestellt.

Was ist eine Activity?

Eine Activity ist eine grafische Oberfläche, die unter anderem die Interaktionsschnittstelle zwischen dem Benutzer und dem Programm darstellt. Es ist nicht möglich mehrere Activities gleichzeitig im Vordergrund anzuzeigen. Wenn eine Activity in den Vordergrund tritt, dann werden alle anderen Activities in den Hintergrund verschoben. Die folgenden Abschnitte sollen verdeutlichen, welche Events vom Betriebssystem zur Verfügung gestellt werden, damit der Entwickler die Möglichkeit hat, auf die verschiedenen Phasen einer Activity zu reagieren. Es wird jedoch nur auf die für das App relevanten Teile des Activity Lifecycle eingegangen.

Verwaltung von Activities - der View-Stack

Da Android bis Version 3.0 nur eine Oberfläche gleichzeitig anzeigen konnte ist es notwendig zu wissen, wie die Activities vom System verwaltet werden.
Activities werden mittels eines Signals vom Betriebssystem geladen. Die zu diesem Zeitpunkt aktive Activity wird pausiert. Auf der Ebene des Betriebssystems werden die Activities in einem Stack verwaltet. Wenn also eine neue Activity gestartet wird, wird sie oben auf den Stack gelegt. Eine Activity kann auch mehrfach unabhängig voneinander gestartet werden. Falls dieser Fall eintritt, existiert die Activity mehrfach in dem Stack. Es ist wichtig diese Mechanik zu verstehen, um ein App mit einer guten Usability zu gewährleisten.

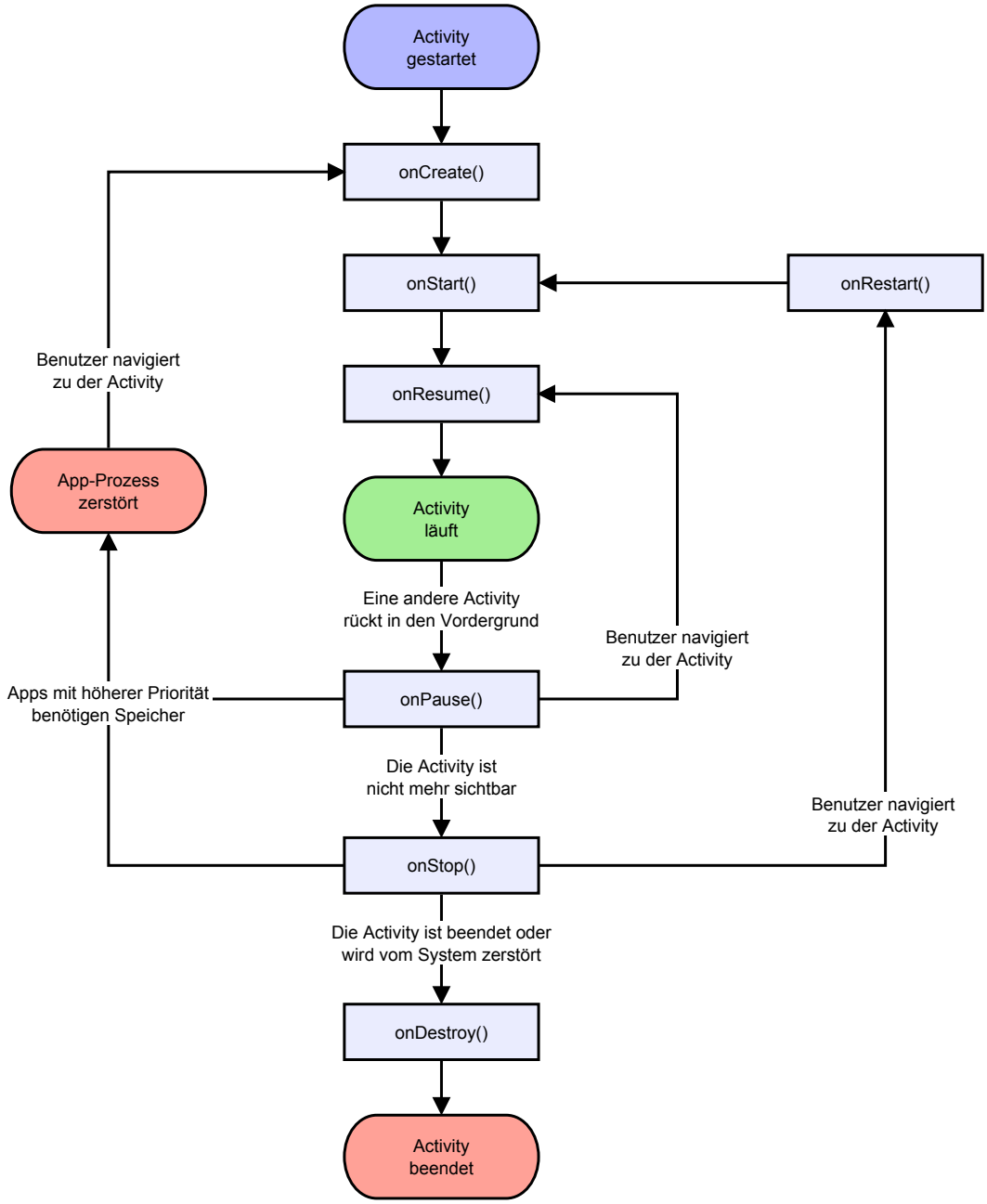


Abbildung 2.3.: Übersicht über den Lebenszyklus einer Activity. Vorlage: [4]
„An Activity is an application component that provides a screen with which users can interact in order to do something [...]. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.“ [3]

Aufrufen einer neuen Activity

Um eine neue Activity zu starten, muss man diese mittels eines so genannten Intents aufrufen. Dieser gibt bekannt, welche Activity in welchem Kontext (i.d.R. der ApplicationContext) aufgerufen werden soll. Dazu muss aus dem Programm der Intent mittels der Methode `startActivity(Intent intent)` an das Betriebssystem übergeben werden. Dies startet die Activity in einem Zustand, der mit einer anonymen Klasse vergleichbar ist. Man hat keine Möglichkeit, irgendwelche Informationen als Callback von der Activity zu erhalten. Falls man dies wünscht, muss man die Activity mittels `startActivityForResult(Intent intent, RequestCode requestCode)` starten. Dabei wird der Activity ein requestCode zugewiesen, mit dem man die Rückgabeinformationen der Activity auswerten kann (dazu später mehr). Nach dem Aufruf der Methode wird die Activity erstellt und tritt in den Vordergrund.

Erstellen einer Activity `onCreate(Bundle bundle)`

Wenn die Activity zum ersten Mal erstellt wird, wird die Methode onCreate(Bundle savedInstanceState) aufgerufen. Diese muss der Entwickler überschreiben, damit er in diesen Schritt eingreifen kann. An dieser Stelle werden für gewöhnlich Objekte instanziiert und gegebenenfalls Variablen auf einen Standardwert gesetzt. Theoretisch ist es möglich, aus dem Parameter Bundle einen vorherigen Zustand der Activity wiederherzustellen. Da dies jedoch im Spiel nicht vorkommt, wird hier nicht weiter darauf eingegangen.

Vorübergehendes pausieren `onPause()`

Wenn die Activity von einer anderen Activity überdeckt wird oder der Benutzer sie minimiert (indem er den Home-Button drückt) wird dies vom System durch das Ausführen der Methode onPause() signalisiert. Hier können beispielsweise Hintergrundprozesse wie Threads oder Services angehalten oder pausiert werden, um die Ressourcen anderweitig nutzbar zu machen.

Wiederaufwachen aus der Pause `onResume()`

Diese Methode ist das Gegenstück zur 'onPause'-Methode. Sie wird ausgeführt, wenn die Activity wieder in den Vordergrund rückt. Diese Methode wird unabhängig davon, ob die Activity erstellt wurde oder aus dem pausierten Zustand wieder fortgesetzt wird, aufgerufen. Zu diesem Zeitpunkt empfiehlt es sich, pausierte Threads fortzusetzen.

Beenden einer Activity `onDestroy()`

Dies ist das Gegenstück zur 'onCreate' -Methode. Sie wird ausgeführt, wenn die Methode `finish()` der Activity aufgerufen wird. Es ist die letzte Möglichkeit für den Entwickler abschließende Aktionen auszuführen, wie Threads zu beenden oder Netzwerkverbindungen zu schließen. Es wird von Google explizit davon abgeraten, zu diesem Zeitpunkt noch Nachrichten im Netzwerk oder innerhalb des Betriebssystems abzusetzen, da nicht gewährleistet werden kann, dass diese noch bearbeitet werden, bevor die Activity vom

Betriebssystem zerstört wird. Innerhalb dieser Methode sollten Ressourcen endgültig freigegeben und gegebenenfalls Rückgabewerte festgelegt werden. Um dies zu tun, holt man sich den Intent der Activity, von der diese Activity gestartet wurde, mittels `getIntent()` und setzt dort so genannte Extras mittels `putExtra(String key, [Object] value)`. Hier kann man auch festlegen, ob die Activity wie beabsichtigt beendet wurde oder ob sie auf anderem Wege beendet wurde indem man bei dem Intent unter setResult ein RESULT_OK oder ein RESULT_CANCELLED angibt.

Weitergabe von Informationen vor dem Beenden einer Activity und deren Auswertung

Sofern die geschlossene Activity mittels `startActivityForResult(...)` aufgerufen wurde, wird nach dem Schließen dieser ein Signal vom Betriebssystem gesendet, welches in der Methode `onActivityResult(int requestCode, int resultCode, Intent data)` abgefragt werden kann.

- Der `requestCode` ist die Variable, die beim Starten der Activity mitgegeben wurde.
- Der `resultCode` entspricht dem resultCode, der von der Activity gesetzt wurde und ist standardmäßig RESULT_CANCELLED.
- `Data` sind alle Daten, die von der Activity mittels `putExtra(...)` gesetzt wurden. Sie können mittels `getExtras()` abgefragt werden.

2.2.6. Einführung in OpenGL ES unter Android

Im folgenden Teil soll dem Leser näher gebracht werden was OpenGL ES ist, worin die Unterschiede zu OpenGL bestehen und warum man ein eigentlich so mächtiges Werkzeug, mit dem normalerweise aufwendige 3D-Spiele entwickelt werden, für eine so unscheinbare Aufgabe wie das Darstellen einer 2D-Grafik in einem Spiel verwendet. Zudem wird erläutert, wie OpenGL ES unter Android verwendet wird und wie der Ablauf der Generierung von Grafiken unter OpenGL, die so genannte Rendering-Pipeline, funktioniert.

Was ist OpenGL (ES)?

„OpenGL® ES is a royalty-free, cross-platform API for full-function 2D and 3D graphics on embedded systems - including consoles, phones, appliances and vehicles. It consists of well-defined subsets of desktop OpenGL, creating a flexible and powerful low-level interface between software and graphics acceleration.“ [12]
OpenGL ist eine plattformunabhängige Grafikbibliothek, die eine Schnittstelle zu Funktionen der Grafikkarte bietet. OpenGL ES ist eine Untermenge dieser Bibliothek, die speziell für mobile Endgeräte entwickelt wurde, da diese häufig Grafikkarten enthalten, die bei weitem nicht so leistungsstark und von der Funktionalität bei weitem nicht so gut ausgerüstet sind wie Desktoprechner. Diese Entwicklung hat sich hauptsächlich ergeben, da mobile Endgeräte energieeffizient arbeiten müssen.

OpenGL sowie OpenGL ES werden vom Khronos Konsortium entwickelt und betreut. Diesem Konsortium gehören unter anderem Intel, AMD, NVIDIA, SGI, Google und Oracle an. Da diese Firmen einen sehr großen Einfluss auf die Entwicklung von Prozessor- und Grafikkartentechnik haben, wird neben OpenGL eigentlich nur noch Direct3D (auch bekannt als DirectX) von Microsoft zur Grafikanzeige mit Grafikkartenunterstützung genutzt.

Die Hauptanwendungsgebiete von OpenGL sind Applikationen, bei denen viele grafische Elemente häufig angezeigt werden müssen. Paradebeispiele für diesen Anwendungsfall sind moderne Computerspiele, bei denen selbst simple Objekte aus tausenden kleinerer Standardobjekte, so genannter Standardprimitive, zusammengesetzt sind. Diese Standardprimitive kann ein Prozessor nicht effizient berechnen. Aus diesem Grund werden Aufgaben wie das Zeichnen von Objekten, die mathematisch einfach sind und sich häufig wiederholen, auf die Grafikkarte ausgelagert, die genau darauf optimiert ist.

Warum OpenGL und nicht direkt auf ein Canvas[9] zeichnen?

Wie schon bei der Definition kurz angedeutet, ist OpenGL ES darauf optimiert, Grafiken häufig anzuzeigen. Es ist dabei nicht von Bedeutung, dass unsere Objekte von sehr einfacher Natur sind. Das Zeichnen auf einem Canvas ist darauf ausgelegt, dass sich das das Canvas nur sehr selten oder gar nicht mehr verändert, wenn es erst erstellt wurde. Da wir in diesem Zusammenhang jedoch von einem Spiel reden ist es offensichtlich, dass wir das Bild sehr häufig neu zeichnen müssen (30 Bilder in der Sekunde wären in Anlehnung an den PAL-Standard erstrebenswert). Aus diesem Grund haben wir uns dafür entschieden, die gesamte Grafikanzeige während des Spiels mit OpenGL ES zu lösen. Aus dem gleichen Grund nutzt auch die Linux-Desktopumgebung *Gnome 3* [1] OpenGL.

Die OpenGL ES-Komponenten unter Android

Die Nutzung von OpenGL ES unter Android funktioniert mittels zweier Komponenten. Diese enthalten drei Events, mit denen man den Zeitpunkt des Erstellens der Oberfläche, einer Veränderung der Konfiguration des Endgerätes und das eigentliche Zeichnen abfängt und implementiert. Diese Struktur wurde gewählt, um den OpenGL-Rendering Prozess vom UI-Thread zu entkoppeln und somit eine gute Reaktionszeit von anderen Elementen wie Buttons oder Ähnlichem zu ermöglichen.

- OpenGLrenderer: Diese Komponente enthält die eigentliche Bibliothek, mit welcher gezeichnet wird. Dabei handelt es sich um ein Interface, das fordert, dass der Entwickler die oben genannten Ereignisse selbst implementiert. Auf die einzelnen Ereignisse wird im nächsten Abschnitt eingegangen.

- GLSurfaceView: Das GLSurfaceView ist die Komponente, welche die Schnittstelle zwischen Oberfläche und OpenGLrenderer herstellt. Sie wird wie jede visuelle Komponente initialisiert und genutzt. Die einzige Besonderheit ist, dass man der

[9] Canvas: Unter einer Canvas versteht man eine Zeichenoberfläche, mit welcher man eine Oberfläche pixelgenau ansteuern kann und dadurch die Pixel färben kann.

Komponente beim Erstellen mitteilen muss, welchen Renderer sie beinhaltet und infolgedessen anzeigt.

Die Renderer-Ereignisse

Wie bereits angedeutet, muss man für den OpenGLrenderer drei Ereignisse implementieren, die hier erläutert werden:

`onSurfaceCreated:` Diese Methode wird einmalig aufgerufen, wenn der Renderer zum ersten Mal erstellt wird. Sie entspricht konzeptionell der onCreate-Methode von Activities. Hier werden interne Variablen initialisiert, Texturen bereitgestellt und Konfigurationsparameter von OpenGL gesetzt.

`onSurfaceChanged:` Normalerweise wird diese Methode aufgerufen, wenn sich die Oberfläche verändert. Dies geschieht, wenn man das Smartphone kippt und es vom Hochformat (dem sogenannten Portraitmodus) in das Querformat (dem Landschafts- oder Landscapemodus) oder umgekehrt wechselt. Da in unserem Spiel der Landschaftsmodus unveränderlich erzwungen wird, sollte diese Methode nur einmal direkt nach der onSurfaceCreated-Methode aufgerufen werden.

`onDrawFrame:` Diese Methode ist die wichtigste der zu implementierenden Methoden, denn sie wird jedes Mal aufgerufen, wenn der Bildschirm neu gezeichnet wird. Es ist wichtig zu beachten, dass es bei einer konsequenten Trennung von Daten und deren Visualisierung vorkommen kann, dass man die Oberfläche nicht neu zeichnen muss, da sich die darzustellenden Daten gar nicht verändert haben.

Rendering Pipeline

Damit mit OpenGL Bilder angezeigt werden können, müssen zunächst mehrere Schritte durchlaufen werden. Diese orientieren sich am Konzept der Grafikpipeline. Zunächst besitzt jedes Objekt ein eigenes absolutes Koordinatensystem. Die Pipeline beschreibt, wie die Koordinatensysteme der Objekte so transformiert werden, dass sie auf einem Display angezeigt werden können. Der genaue Aufbau der Pipeline kann Abbildung 2.4 entnommen werden. Die entsprechenden Matrizen kann man selbst anlegen und in die Renderpipeline einfügen. Unter OpenGL ES 1.1 gibt es zudem die Möglichkeit, die Model-Matrix und die View-Matrix direkt zu kombinieren und dann direkt mir der so genannten Model-View-Matrix zu arbeiten [16].

Wir sind jedoch noch einen anderen Weg gegangen und haben die GLU[10] verwendet, die Methoden zur Verfügung stellt, mit denen die Renderpipeline automatisch vorbereitet wird.

[10] GLU: OpenGL Utility Library (engl.)

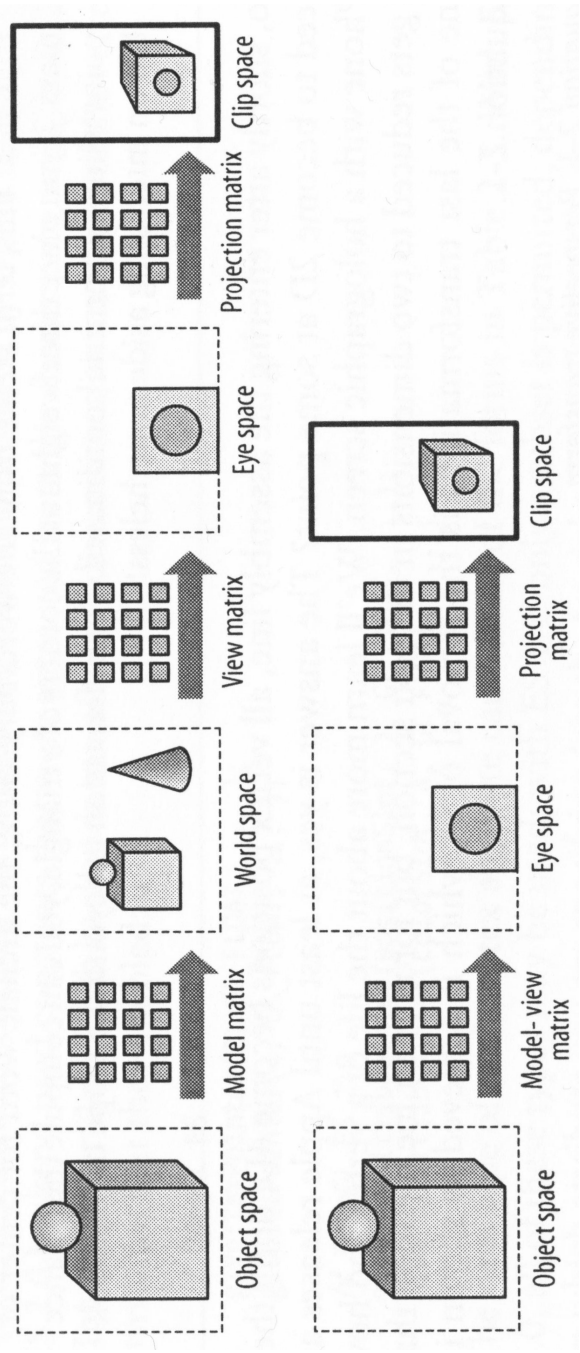


Abbildung 2.4.: Visuelle Darstellung der Grafikpipeline [16, Seite 50]

3. Konzept

3.1. Architektur des Spiels

Im Folgenden soll die Architektur des Apps vorgestellt werden. Der Aufbau ist nach den einzelnen Komponenten des Spiels gegliedert. Zunächst wird der generelle Aufbau erläutert, woraufhin auf die genutzten Konzepte eingegangen wird. Danach werden die einzelnen Komponenten der verwendeten Design-Patterns vorgestellt. Dabei werden auch Komponenten aus der Arbeit von Michael Merg vorkommen. Diese werden zwar vorgestellt, es wird jedoch nicht näher auf deren Implementierung im Spiel eingegangen.

3.1.1. Aufbau

Grundsätzlich lässt sich das Programm in mehrere Untergebiete aufteilen, die verschiedene Komponenten und gegebenenfalls unterschiedliche Design-Patterns nutzen.

Menü: Es gibt eine Menüführung, die mittels untertitelten ImageButtons den Benutzer durch das „Setup“ führt, in dem der Modus sowie die gegebene Strecke und alle Stellparameter ausgewählt werden können. In manchen Modi lassen sich zusätzliche Informationen über die Auswirkungen der Stellparameter visualisieren. Die Grundlagen zu den entsprechenden Parametern finden sich in Anhang A. Man kann sich zudem in den meisten Menüs durch einen Hilfebutton anzeigen lassen, was die verschiedenen Auswahlmöglichkeiten bewirken. Vom Hauptmenü aus sind des weiteren der Highscore und die Einstellungen abrufbar.

Highscore: Der Highscore soll tabellarisch die besten zehn Spieler anzeigen. Hierzu wurde als Bewertungsgrundlage der Leistung die Abweichung des Raumschiffes von der vorgegebenen Strecke gewählt. Eine nähere Beschreibung findet sich im Kapitel 3.1.8 auf Seite 25.

Einstellungen: Dem Benutzer werden hier verschiedene Einstellungsmöglichkeiten zur Individualisierung und zur besseren Handhabung und Identifikation mit dem Spiel zur Verfügung gestellt. Hier kann man seinen eigenen Namen angeben (der später im Highscore angezeigt wird) und man kann angeben, wie sich das kippen des Smartphones auf die Beschleunigung des Raumschiffes im Spiel auswirken soll. Hier wird zwischen logarithmischem, exponentiellem und linearem Verhalten unterschieden.

Spieloberfläche: Das eigentliche Spiel ist intern durch ein MVC[1] Design Pattern implementiert. Dabei ist das Datenmodell eine Implementierung des Reglers, der in A beschrieben wird. Der View ist der Hauptbestandteil dieser Arbeit und wird durch

[1] MVC: Model View Controller

einen OpenGL-Renderer umgesetzt. Da beide Elemente in der eigentlichen Activity enthalten sind, kann man diese als Controller betrachten. Das Konzept wurde eigentlich für normale Programmiersprachen entworfen, die nicht schon als Basisklasse eine grafische Oberfläche darstellen. Wir haben uns daher möglichst nah an der Idee des MVC Patterns gehalten, auch wenn dies im strikteren Sinn kein MVC mehr ist. Hier wurden außerdem Konzepte der Spielentwicklung, wie eine zeitdiskrete Darstellung oder eine direkte Rückmeldung von Eingaben verstärkt berücksichtigt.

Zusammenfassungsbildschirm: Hier wird das Abschneiden während eines Spiels noch einmal zusammengefasst dargestellt. Es gibt eine Visualisierung über die Abweichung von der idealen Strecke und die Punktzahl. Außerdem kann der Spieler noch einmal all seine Stellparameter, sowie seine Punktzahl und den Namen ansehen.

3.1.2. Genutzte Konzepte und Design-Patterns

Konzepte und Design-Patterns wurden entwickelt, um die Übersicht und die Wartbarkeit von Programmen zu erhöhen und Anwendungserweiterungen potenziell einfacher umsetzbar zu machen. In diesem Abschnitt wird daher auf die verschiedenen Konzepte und Design-Patterns eingegangen, die im Spiel genutzt wurden. Die meisten genutzten Konzepte wurden im eigentlichen Spiel genutzt. Im Folgenden sei der Begriff 'Spiel' die eigentliche Spielkomponente.

Konzept für die intuitive Nutzbarkeit von Spielen

Bereits im Jahr 2003 veröffentlichte Nokia grundlegende Richtlinien für Spiele auf mobilen Geräten mit dem Titel „Nokia Series 40 J2ME Game Usability Guidlines and Implementation Model“. Hierin wurden 10 Punkte aufgelistet, die bei einem Spiel durch erhöhte Nutzbarkeit für eine erhöhte Akzeptanz sorgt. Obwohl einige der Punkte entweder auf unser Projekt nicht zutreffen oder bereits veraltet sind, werden diese hier kurz aufgelistet und gegebenenfalls deren Umsetzung angesprochen:

1. Für eine klare Menüstruktur sorgen: Obwohl dieser Punkt eigentlich selbstverständliche sein sollte haben wir versucht, dem Benutzer unseres Spiels eine möglichst einfache und klar strukturierte Menünutzung zu ermöglichen.

2. Einfachheit ist der Schlüssel zum Erfolg: Falls es zwei Wege gibt, dem Benutzer eine Lösung eines Problems zu ermöglichen, sollte man immer die einfachere zur Verfügung stellen. In unserem Fall hätte man zum Beispiel überlegen können, ob man immer einen Modus mit und einen Modus ohne Einfluss des Spielers erstellt (also regelt zum einen ein Regler das Raumschiff, zum anderen kann der Spieler jedoch auch selbst noch Einfluss auf die Beschleunigung nehmen). Wir haben hiervon jedoch abgesehen und dem Spieler beim Einstellen der Parameter die Möglichkeit zur Verfügung gestellt, durch eine Checkbox den „Hybrid-Modus“ zu aktivieren, welcher genau diesen Einfluss des Spielers selbst ermöglicht.

3. Hilfe anbieten, wenn sie benötigt wird: Die meisten unserer Menüs haben einen kleinen Hilfeknopf, mit dem man sich eine Beschreibung der verschiedenen gerade sichtbaren Funktionen anzeigen lassen kann. Dies ist jedoch rein optional. Wenn ein Spieler der Meinung ist, er braucht keine Hilfe, kann er gänzlich auf die Hilfestellungen verzichten.

4. Unerbittliche Konsequenz zeigen: Obgleich es keine Lokalisierung für das Spiel gibt, so sind doch alle Texte durchgehend auf Englisch. Gebräuchliche Eingabemethoden wie eine Auswahl per Toucheingabe und eine Rückgängig-Funktion mittels der Rückgängig-Taste wurden durchgehend umgesetzt.

5. Nicht die Zeit des Benutzers verschwenden: Alle Einstellungsdaten werden vom Programm intern gespeichert und müssen nicht bei jedem Programmstart neu eingegeben werden. Die Parameter in den einzelnen Reglerkonzepten setzen sich jedes Mal zurück. Da das Spiel einen Lerneffekt haben soll und man die korrekte Lösung errechnen kann wäre es nicht sinnvoll, vom Programm aus gute Werte vorzugeben oder die Werte zu speichern.

6. Gegebene Steuerung nutzen: Diese Richtlinie ist veraltet (es gibt keine Zahlenfelder mehr und es ist nicht sinnvoll, solche Funktionen zu emulieren).

7. Speichern und Pausieren ermöglichen: Da das Spiel ohnehin nur bis zu einer Minute geht, ist es nicht sinnvoll, dem Benutzer das Speichern des Spieles zu ermöglichen. Wir gehen davon aus, dass man das Spiel dann einfach neu startet.

8. Den Erwartungen der Wirklichkeit entsprechen: Die wenigsten (vermutlich keine) Studenten werden jemals in einem Raumschiff geflogen sein und sich mit deren Beschleunigungsverhalten auskennen. Wir haben uns daher entschieden, dem Raumschiff ein Beschleunigungsverhalten zu geben, dass Masse, Trägheit, Gravitation stark vereinfacht. Reibung, Trägheut und Gravitation werden durch einen Doppelintegrator beschrieben und die Reibung wird ignoriert.

9. Nicht übertreiben bei den Soundeffekten: Da unser App keine Soundeffekte enthält stellt sich die Problematik einer angemessenen Beschallung des Benutzers nicht.

10. Eine Highscoreliste implementieren: Es existiert eine Highscore-Liste und diese wird in dieser Arbeit auch in einem extra Kapitel behandelt.

Konzept: Zeitdiskretisierung im Spiel

Es stellt sich in vielen Spielen, in denen mathematische Funktionen jedweder Art vorkommen, ein grundsätzliches Problem: Funktionen, die kontinuierlich sind, sind es in Spielen irgendwann nicht mehr. Egal wie man das Spielkonzept entwickelt, es tritt immer irgendwann das Problem auf, dass man das Spiel spätestens aufgrund der Bildrate der Anzeige bezüglich der Zeit diskretisieren muss. Es wäre schließlich weder im Interesse des Entwicklers noch der des Anwenders, wenn ein Spiel auf unterschiedlichen Endgeräten unterschiedlich schnell oder langsam abläuft, oder die Steuerung direkter oder verzögert

reagiert. Für den Entwickler stellt sich dann das Problem, dass er das Programm nicht sinnvoll optimieren kann (weder bei der Leistung noch bezüglich des Spielspaßes) und für den Anwender ergibt sich ein nicht vorhersehbares Spielerlebnis. Dies beeinflusst im Normalfall den Spielspaß und die Verwendungshäufigkeit negativ.

In unserem Spiel ergibt sich die Diskretisierungsproblematik noch an anderen Stellen. Zum einen muss der Regler beim Integrator eine Diskretisierung vornehmen, welche die mathematische Genauigkeit des Reglers beeinflusst. Daraus folgt auch, dass der Regler an sich diskretisiert wird. Dies stellt ein Problem bezüglich der Synchronisation von Regler und Sensoreingabe dar. Als weitere Konsequenz muss man sich auf Seite der Visualisierung Gedanken darüber machen, wann es überhaupt Sinn macht, die Daten darzustellen, da diese eventuell seltener aktualisiert werden, als das Bild gezeichnet wird. Zum anderen ist es nicht möglich, eine unbegrenzt hohe Anzahl an Bildern pro Sekunde auf einem Display darzustellen. Dies wird durch die reinen hardwaretechnischen Möglichkeiten begrenzt. Dies hat zur Folge, dass man nicht nur auf Seiten des Datenmodells Einschränkungen bezüglich der Aktualisierungsrate hat, sondern auch auf der Seite der Anzeige.

Die Visualisierungsebene muss zudem berücksichtigen, dass das menschliche Auge je nach Bewegungsgeschwindigkeit der Umgebung bei langsamen Bewegungen zwar nur 12-14 Bilder pro Sekunde benötigt, um eine Bildfolge als flüssige Animation wahrzunehmen. Unser Spiel enthält jedoch schnellere Bewegungen, weshalb wir eine höhere Bildwiederholungsrate anzustreben. Diese darf jedoch nicht zu hoch sein, da eine hohe Bildwiederholungsrate viel Rechenzeit benötigt. Eine optimale Framerate von 30 FPS[2] scheint sinnvoll, da diese Framerate bereits beim PAL-Standard für europäische Fernseher verwendet wurde. Dies wird im Optimierungskapitel auch zur Schonung der Ressourcen genutzt. Die Problematik besteht darin, dass die Anzeige nicht einfach annehmen kann, dass die vorgenommene zeitliche Diskretisierung der Anzeige (die optimalerweise 30 FPS ist) eingehalten werden kann. Dies gilt sowohl für den Zeichenvorgang (falls keine Ressourcen zur Verfügung stehen) als auch für den Aktualisierungsprozess des Reglers und des Raumschiffes. Hierdurch kann es zu ausgelassenen Bildern kommen, die im schlimmsten Fall durch eine nicht flüssige Animation wahrgenommen werden. Allein die Annahme eines verhältnismäßig langsamen Smartphones, das eventuell die gewünschte Bildrate nicht erreicht, sollte als Argument ausreichen, um sich Gedanken über eine sinnvolle Schnittstelle zwischen Regler und Visualisierung zu machen.

Design-Pattern MVC

Das Model-View-Controller (kurz MVC) Entwurfsmuster aus der Softwaretechnik beschreibt ein Muster zur Entkopplung des Datenmodells von der grafischen Repräsentation. Als Kommunikationsschnittstelle dient dabei ein Controller, der gleichzeitig als Interaktionsschnittstelle fungiert. In unserem Spiel ist das Datenmodell das Raumschiff und die Visualisierung ist die OpenGL-Oberfläche. Als Controller soll die Activity dienen, die sowohl das Datenmodell als auch die grafische Oberfläche enthält.

[2] FPS: frames per second (engl.) $\hat{=}$ Bilder pro Sekunde (dt.)

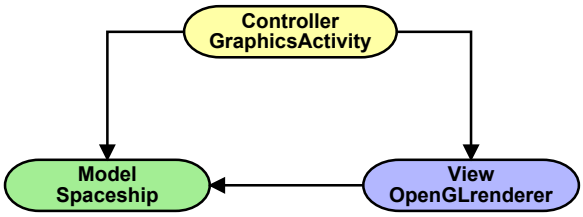


Abbildung 3.1.: Schematische Darstellung des MVC

3.1.3. Controller

Der Controller ist wie bereits beschrieben das Bindeglied zwischen Datenmodell und Visualisierung. In unserem Fall ist der Controller die Activity, welche die OpenGL-Oberfläche enthält. Ihre Funktionsweise ist nur an die Idee der MVC-Controllerklasse angelegt. In unserem Fall stellt sie hauptsächlich den Container dar, der beide Klassen enthält und übernimmt selbst sogar die Visualisierung des HUD[3].

3.1.4. Sensor

Um dem Benutzer eine Möglichkeit zu geben mit dem Raumschiff zu interagieren ist es nötig die Sensoren des Smartphones anzusteuern und deren Daten zu verarbeiten und auszuwerten. Auf den meisten Smartphones stehen bezüglich der Lokalisierung des Smartphones zumindest ein Lagesensor, ein Beschleunigungssensor und ein Kompass zur Verfügung. Für unsere Zwecke ist jedoch der Lagesensor ausreichend, da die Beschleunigung des Raumschiffes durch die Kippneigung des Smartphones beeinflusst werden soll, welche einer der Freiheitsgrade des Lagesensors ist. Hierdurch wird es möglich den Menschen in den Regelkreis zu integrieren.

Abbildung 3.2.: Konzept zur Integration einer menschlichen Komponente in den Regelkreis

Die genauere Beschreibung des Sensors kann der Arbeit von Michael Merg [13] entnommen werden.

[3]HUD: Head-up Display (engl./dt.) - Ein Anzeigesystem, bei dem die für den Nutzer wichtigen Informationen in sein Haupt-Sichtfeld projiziert werden, so dass er seine Kopfhaltung bzw. Blickrichtung kaum ändern muss [21].

3.1.5. Menü

Normalerweise empfiehlt es sich , dem Anwender ein Menü im Stil der Umgebung des Betriebssystems zur Verfügung zu stellen, damit das App besser mit dem Betriebssystem harmoniert. Es gibt jedoch Fälle, bei denen diese Stilrichtlinie nicht zutrifft. Zum einen kann man bei Smartphone-Apps eine zunehmende Differenzierung der verschiedenen Apps über ihre Menüführung und ihren Stil beobachten. Dies geschieht hauptsächlich, damit sich die einzelnen Apps von ihrer Konkurrenz unterscheiden, da die meisten Programme des gleichen Metiers eine sehr ähnliche oder sogar die gleiche Funktionalität zur Verfügung stellen. Zum anderen ist diese Richtlinie bei Spielen nicht wirksam, da Spiele ein in sich geschlossenes System darstellen sollten, das Betriebssystem also nicht in Erscheinung treten sollte. Das Menü des Spiels soll dem Benutzer die Funktionen des Apps möglichst einfach und intuitiv zur Verfügung stellen, ohne dabei den Benutzer mit einer Flut von Informationen zu verwirren oder im schlimmsten Fall abzuschrecken.

Aus diesem Grund werden die meisten Funktionen dem Benutzer nur zugänglich gemacht, wenn dies auch sinnvoll ist.

- Einstellungen wie der Spielername oder die Art, wie der Sensor auf das Spiel Einfluss haben soll, können vom Titelbildschirm über das Optionsmenü eingestellt werden.
- Der Spielmodus ist die erste Auswahl, die der Benutzer treffen muss. Je nach Auswahl wird entweder ein Regler mittels Zustandsrückführung, eines zusätzlichen Beobachters oder Loop-Shapings genutzt, um das Raumschiff zu regeln.
- Nachdem der Spielmodus gewählt wurde, können abhängig vom Modus diverse Parameter gewählt werden, die meisten der Parameter werden in dieser Studienarbeit nicht genauer behandelt:
  - Der Hybrid-Modus kann, abgesehen vom Sensormodus (in welchem es nur die Benutzereingabe gibt), in jedem Modus optional hinzugeschaltet werden.
  - Sollte der Benutzer den Sensor ansteuern (d.h. im Sensor-Modus oder im Hybrid-Modus sein), kann er mittels des *Acceleration*-Parameters die Stärke des Sensors auswählen. Diese Verstärkung des Sensors wird unabhängig vom Modus mit dem Sensorwert multipliziert.
  - Falls die Zustandsrückführung mit oder ohne Beobachter ausgewählt wurde, muss man die k_1- und k_2-Parameter für den Regler einstellen.
  - In Fall des Beobachters müssen des Weiteren der l_1- und der l_2-Parameter angegeben werden.
  - Beim Loop-Shaping-Modus müssen die Werte für k_p, v und m angegeben werden.
- Zusätzlich zu den Parametern können in manchen Modi zusätzliche Visualisierungen aufgerufen werden.
  - Die Modi für Regler mit einer Zustandsrückführung mit oder ohne Beobachter können Pol-Diagramme anzeigen.
  - Das Loop-Shaping unterstützt die Visualisierung eines Bode-Diagramms.

3.1.6. Grafische Oberfläche

Die grafische Repräsentation des Spiels ist der Fokus dieser Arbeit. Im Folgenden wird daher genauer auf die einzelnen Komponenten eingegangen und wie sie Informationen repräsentieren beziehungsweise die User Experience erhöhen.

Die vorgegebene Trajektorie

Damit der Spieler eine Vorstellung von der Trajektorie hat, der er folgen soll, muss eine visuelle Repräsentation dieser Trajektorie gegeben sein. Die Funktion, welche die Trajektorie repräsentiert wurde zuvor vom Benutzer ausgewählt und wird vom Datenmodell des Raumschiffes verwaltet. Diese Funktion ist kontinuierlich und muss daher diskretisiert werden, da Computer nicht in der Lage sind, kontinuierliche Funktionen zu zeichnen. Dies würde jedoch eine infinitesimale Taktzeit erfordern, was physikalisch nicht möglich ist. Nachdem die Trajektorie diskretisiert wurde muss diese grafisch angezeigt werden. Man kann die Auswertungen der Funktion an den einzelnen Stellen als Punktmenge ansehen. Da a priori bekannt ist, dass die x-Werte der Punkte streng monoton wachsend sind bietet sich eine Repräsentation über Liniensegmente an.

Das Raumschiff

Neben dem Datenmodell des Raumschiffs (welches den Regler enthält) muss es eine grafische Repräsentation dieser Daten geben, die folgende Daten repräsentiert:

1. *Die aktuelle Position:* Der Spieler muss wissen, auf welcher Höhe er fliegt. Dies würde dem y-Wert im Datenmodell entsprechen.
2. *Die aktuelle Flugrichtung:* Da der Spieler beim Kippen des Smartphones nicht die Geschwindigkeit, sondern die Beschleunigung des Raumschiffes beeinflusst ist es notwendig, die aktuelle Beschleunigung bzw. Ausrichtung des Raumschiffes zu repräsentieren. Dies lässt sich am einfachsten mittels einer Rotation des Raumschiffes bewerkstelligen.

Bewegte Sterne

Es soll eine fixe Anzahl an Sternen geben, die dem Benutzer angezeigt werden. Dies soll die Umwelt des Spiels authentischer wirken lassen. Die Sterne sollten sich mit unterschiedlicher Geschwindigkeit bewegen und unterschiedlich groß sein, um dem Spieler eine Gefühl der räumlichen Darstellung zu geben, obwohl sich das Spiel auf einer zweidimensionalen Ebene abspielt. Des Weiteren macht es Sinn die Sterne dynamisch zu generieren, damit der Spieler nicht das Gefühl bekommt, dass sich die Umgebung ständig wiederholt, was eine erhöhte Immersion zur Folge hat.

Hintergrund

Wir haben uns entschieden, nur die Sterne prozedural zu generieren. Um die Spielatmosphäre weiter zu verbessern haben wir uns entschieden, passend zu der auch sonst

recht einfach gehaltenen Grafik, ein Bild als Hintergrund zu verwenden. Die beste Quelle für gute Weltraumbilder sind die Raumfahrtbehörden der Welt selbst. Die NASA stellt Weltraumaufnahmen im Internet zur freien Verwendung zur Verfügung. Dabei haben wir darauf geachtet ein Bild zu verwenden, bei dem man den Hintergrund noch deutlich von den anderen Spielelementen unterscheiden kann.

Verlauf des eigenen Flugs

Um dem Spieler ein erstes Feedback über sein bisheriges Abschneiden zu geben macht es Sinn, die geflogene Strecke direkt zu visualisieren. Diese Visualisierung sollte sich von der Visualisierung der vorgegebenen Strecke dahingehend unterscheiden, als dass der Benutzer eindeutig erkennen sollte, wie gut er spielt bzw. wie gut der Regler funktioniert. Auch hier macht es (vor allem aus Auslastungsgründen) Sinn, sich Gedanken für die Diskretisierung der Daten des Raumschiffes zu machen.

Positionierung des Raumschiffs

Man muss sich bezüglich der Positionierung des Raumschiffs Gedanken darüber machen, ob das Raumschiff in der Mitte des Displays gezeichnet werden soll oder ob und wie weit man es an den linken Rand des Displays verschiebt. Die Gegenüberstellung der Vor- und Nachteile der verschiedenen Positionierungen können Tabelle 3.1 entnommen werden.

Position	Vorbereitungszeit	Rückschau
Linker Rand	Bietet die längste Vorschauzeit	So gut wie kein Feedback über eigenes Abschneiden
Linker Rand mit Puffer	Lange Vorschauzeit	Kurze Rückschau auf eigenen Verlauf
Mittig	Kurze Vorschauzeit	Lange Rückschau auf eigenen Verlauf

Tabelle 3.1.: Denkbare Positionierungen des Raumschiffs

Es scheint sinnvoll das Raumschiff nicht zentral, sondern leicht nach links verschoben anzuzeigen, da der Benutzer vermutlich mehr Nutzen aus der Information ziehen kann, was in näherer Zukunft gefordert sein wird als was bereits passiert ist. Die Informationen aus der Rückschau sind jedoch trotzdem nützlich, da der Benutzer damit schon während des Spiels evaluieren kann, wie er sein Verhalten anpassen muss, um sein restliches Abschneiden zu verbessern.

Synchronisation der Trajektorien und des Raumschiffes

Ein sowohl für die User Experience als auch für die Batterieauslastung und das Punktbewertungssystem elementares Problem ist die Synchronisation der Diskretisierung der Visualisierung der einzelnen Komponenten mit deren Datenmodell. Die genaue Diskretisierung des Datenmodells ist Gegenstand der Studienarbeit von Michael Merg.

Head-up Display

Unter einem Head-up Display versteht man eine Ansicht, die die ansonsten angezeigte Visualisierung überlagert und wichtige Informationen anzeigt. In unserem Fall gehört zu diesen Informationen

- Die aktuelle Punktzahl und
- Die aktuelle Spielzeit.

Diese Informationen helfen dem Spieler, sein eigenes Abschneiden zu jedem Zeitpunkt des Spiels einzuschätzen.

3.1.7. Zusammenfassungsbildschirm

Der Zusammenfassungsbildschirm soll den Spieler durch eine Zusammenfassung seines Abschneidens in die Lage versetzen, seine Ergebnisse selbst noch einmal zu evaluieren und gegebenenfalls Anpassungen an den Reglerparametern vorzunehmen. Um dies zu gewährleisten sollten dem Spieler folgende Informationen zur Verfügung gestellt werden:

Reglerparameter: Da nicht zu erwarten ist, dass sich der Spieler für die gesamte Zeit des Spiels die eingestellten Reglerparameter merkt, sollten diese noch einmal aufgelistet werden.

Verlauf: Da der Spieler während des Spiels immer nur einen kleinen Ausschnitt der Trajektorie und seiner Leistung sehen kann haben wir uns entschieden, die beiden Trajektorien noch einmal als Gesamtverlauf anzuzeigen.

3.1.8. Highscore

Zweck eines Highscores

Der Hauptzweck eines Highscores ist die Aufrechterhaltung der Langzeitmotivation für den Spieler. Dazu müssen mindestens

- der Name des Spielers und
- die erreichte Punktzahl

angezeigt werden.

Erweiterungen unseres Highscores

Da es in unserem Fall zudem verschiedene Modi gibt ist es sinnvoll, nicht einen Highscore für alle Spiele, sondern für jeden Modus einen eigenen Highscore anzulegen. Außerdem unterscheiden sich die auswählbaren Trajektorien bezüglich ihrer Komplexität stark, was der Spieler als höhere Schwierigkeit wahrnimmt und die verschiedenen Trajektorien schwer vergleichbar macht.

Maximale Punktzahl und Metrik zur Punktberechnung

Um dem Spieler eine klarere Vorstellung davon zu geben, was die perfekte Punktzahl ist, halten wir sowohl die maximale Punktzahl als auch die Berechnung der eigenen Punktzahl sehr einfach. Die maximale Punktzahl berechnet sich aus der Formel $\text{Punktzahl} = \text{Rundenlänge[s]} \cdot \frac{\text{Messungen}}{\text{Sekunde}} \cdot 10$ wobei 10 die maximale erreichbare Punktzahl pro Messung ist.

4. Implementierung

Dieses Kapitel befasst sich mit der konkreten Implementierung des Spiels. Die Visualisierung der Systemarchitektur findet sich in 4.1. Eine vollständige Fassung der Architektur des Programms findet sich als UML-angelehntes Diagramm in Anhang B. Zunächst werden die verschiedenen Phasen des Spiels näher erläutert. Daraufhin wird näher auf die konkrete Implementierung der grafischen Visualisierung des Spiels eingegangen. Dabei werden sowohl die sichtbaren Elemente erläutert als auch Techniken zur Leistungsoptimierung. Anschließend wird näher auf die Schnittstellen zwischen Modell und Visualisierung eingegangen. Im Verlauf dessen werden die konzeptionellen Probleme der Schnittstelle zwischen dem Modell und der Visualisierung erläutert. Im Anschluss werden die Vor- und Nachteile von verschiedenen Lösungsansätzen aufgezählt und es wird erläutert, welchen Lösungsansatz wir gewählt haben und warum. Zuletzt wird auf die Implementierung des Highscores eingegangen.

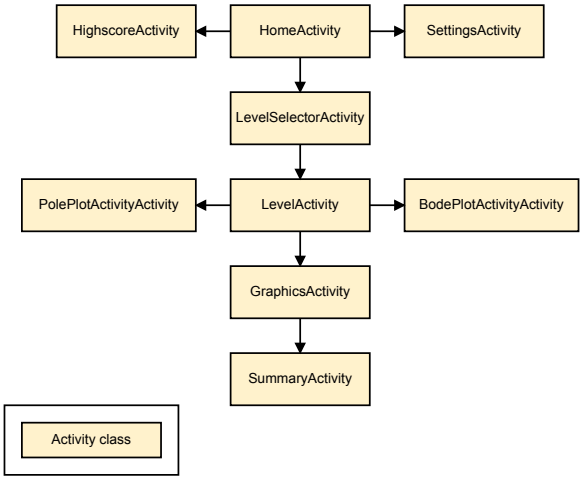


Abbildung 4.1.: Kurzfassung der Systemarchitektur des Spiels

4.1. Spielphasen

Das Spiel unterteilt sich in verschiedene Phasen. So muss vor Beginn des eigentlichen Spiels festgelegt werden, in welchem Modus man das Spiel spielen möchte und wie die benötigten Parameter gewählt werden. Der folgende Abschnitt befasst sich mit den unterschiedlichen Spielphasen.

4.1.1. Auswahl des Spielmodus

Zu Beginn muss der Spieler einen Spielmodus auswählen. Dabei stehen vier verschiedene Modi zur Auswahl. Diese werden in Tabelle 4.1 aufgelistet. Die Modi werden durch ImageButtons dargestellt, zwischen denen man mit einer Wischgeste wechseln kann.

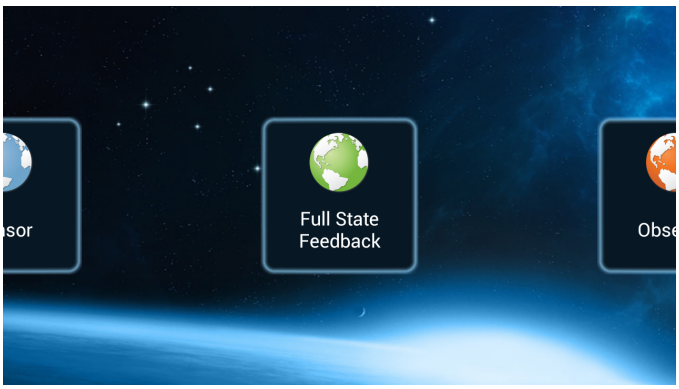


Abbildung 4.2.: Screenshot des Bildschirms zur Auswahl des Spielmodus

4.1.2. Festlegung der Parameter des Reglers

Nachdem der Anwender den Spielmodus ausgewählt hat muss er im nächsten Bildschirm die Parameter einstellen, die der zuvor gewählte Regler benötigt. Diese Parameter haben wir in Anlehnung an die Numberpicker des Betriebssystems nachgebaut, da sie von der Android 2.2-API nicht zur Verfügung gestellt werden. Die Numberpicker sind dabei ein Container, der folgende Elemente enthält:

- Einen TextView für den Titel des NumberPickers.
- Zwei ImageButtons um eine Erhöhung bzw. Reduzierung des angezeigten Wertes zu ermöglichen.
- Einen EditText um den Wert darzustellen.

Abbildung 4.3.: Abbildung unserer Implementierung für die Auswahl der Parameter des Reglers bzw. des Sensors. Konkret dargestellt ist die Komponente, mit der man die Verstärkung des Sensors wählen kann.

Modus	Erforderliche Kenntnis in Regelungstechnik	Vorteile	Nachteile
Sensor	Keine	Einfach zu verwenden; Kein Rauschen	Viele Spieler haben anfangs einen Hang zum Überschwingen; Keinerlei Hilfe durch einen Regler
Full State Feedback (Strecke: $\frac{1}{s^2}$)	Grundlegende Kenntnis im Gebiet der Regelungstechnik	Wenige Parameter; Einfach zu berechnen; Abwägung zwischen Rückkopplungsstärke und Robustheit	Rauschen; Kann instabil werden; Abwägung zwischen Rückkopplungsstärke und Robustheit
Beobachter (Strecke: $\frac{1}{s^2}$)	Funktionsweise des Full State Feedbackreglers und der Beobachter-Komponente	Funktioniert mit weniger Informationen; Alle Vorteile des Full State Feedback-Reglers	Etwas aufwändiger zu berechnen (Zwei zusätzliche Parameter); Alle sonstigen Nachteile des Full State Feedback-Reglers
Loop-shaping (Strecke: $\frac{1}{s^2}$)	Zusammenhang von Zeit- und Frequenzbereich; Interpretation von Bode-Diagrammen	Kann viele Parameter verarbeiten; Funktioniert mit Optimierungsprogrammen; Arbeitet im Frequenzbereich	Kann transiente Phase nicht betrachten; Nur schlecht manuell optimierbar

Tabelle 4.1.: Auflistung der Spielmodi mit Vor- und Nachteilen

Abbildung 4.4.: Screenshot der Poldiagramms. Für eine verbesserte Lesbarkeit wurden die Farben invertiert

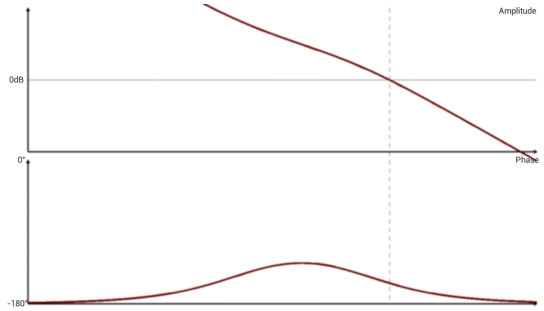


Abbildung 4.5.: Screenshot des Bodediagramms. Für eine verbesserte Lesbarkeit wurden die Farben geändert.

Nachdem alle Parameter eingestellt wurden kann der Benutzer mit dem „Weiter"-Knopf das Spiel starten. Beim Starten des Spiels werden die Parameter aus den NumberPicker-Elementen ausgelesen und als Parameter an die Spiel-Activity übergeben, die dann die Parameter über dessen Konstruktor an dessen Datenmodell weitergibt.

4.1.3. Optional: Anzeige von Pol- und Bodediagramm

Mit Ausnahme des Sensor-Modus kann man sich in allen anderen Modi je eine Visualisierung anzeigen lassen. Im Falle des Zustandreglers (mit und ohne Beobachter) kann man sich ein Pol-Diagramm anzeigen lassen (siehe Appendix A.2). Die Interaktionsmöglichkeiten mit dem Diagramm und dessen Auswirkungen sind nicht Teil dieser Arbeit. Beim Loop-Shaping kann ein Bode-Diagramm angezeigt werden (siehe Appendix A.4)

Beide Visualisierungen sind zur Unterstützung des Benutzers beim Wählen der Parameter gedacht und sind vollkommen optional. Sie können unbegrenzt oft aufgerufen werden.

4.1.4. Spiel

Das Spiel ist wie im Konzept beschrieben in drei Komponenten aufgeteilt. Die Activity dient als Container für das Datenmodell des Raumschiffs, der OpenGL-Visualisierungskomponente und der Timer, die Spielbeginn und Spielende definieren und sie stellt das HUD zur Verfügung. Das Datenmodell berechnet (ggf. unter Zuhilfenahme des Lagesensors) die Position des Raumschiffs, dessen Abweichung zur Soll-Trajektorie und verteilt dementsprechend Punkte für den Highscore. Die Visualisierungskomponente wird in einem eigenen Abschnitt ausführlich besprochen. Ihre Funktion ist die visuelle Darstellung der Daten aus der Raumschiff-Klasse.

4.1.5. Spielzusammenfassung

Um dem Spieler eine bessere Übersicht über sein Abschneiden zu gewährleisten, wird ihm nach Spielende eine Zusammenfassung über sein Abschneiden angezeigt.

Metainformationen

Im oberen linken Abschnitt werden Spielername, erreichte Punktzahl und der damit einhergehende Platz im Highscore des entsprechenden Modus mit der entsprechenden Vorgabefunktion angezeigt. Der Name wird aus den App-internen Einstellungen ausgelesen. Die Punktzahl wird aus als Extra-Information an die Activity übergeben. Der erreichte Platz wurde mittels einer Datenbankabfrage an die Highscore-Datenbank realisiert.

Regler-Informationen

Im oberen rechten Abschnitt werden dem Spieler noch einmal die zu Beginn des Spiels gewählten Reglerparameter angezeigt. Diese werden (genau wie die zuvor genannte Punktzahl) aus den von der aufrufenden Activity übergebenen Extra-Informationen ausgelesen und sind kontextabhängig. Es werden also nur die Parameter angezeigt, die auch tatsächlich für den entsprechenden Regler benötigt wurden.

Flugbahnzusammenfassung

Der untere Abschnitt zeigt eine Übersicht über den gesamten Verlauf des Spiels. Die Trajektorien von Raumschiff und Vorgabefunktion werden gleichzeitig gegen die Zeit aufgetragen.

4.1.6. Highscore

Nachdem das Spiel beendet wurde, wird dem Spieler mittels eines Toasts[1] die erreichte Punktzahl anzeigt. Außerdem wird die Highscore-Activity aufgerufen. Diese besteht aus zwei Auswahlkomponenten mit denen man den Spielmodus und die Soll-Trajektorie ausgewählt werden kann. Darunter werden in einer Listenansicht die besten zehn Spieler mit Namen und erreichter Punktzahl angezeigt. Es ist kein Problem die Auswahlkomponenten zur Verfügung zu stellen. Da wir jedoch in der Liste den Spielernamen linksbündig und seine erreichte Punktzahl rechtsbündig in der selben Zeile anzeigen wollten mussten wir eine Anpassung an der ListView vornehmen, da diese die von uns übergebenen Informationen untereinander angezeigt hätte. Um unser gewünschtes Layout zu erreichen haben wir selbst mittels eines XML-Layouts die Darstellung eines Listeneintrags definiert und haben diese Darstellung der entsprechenden Schnittstelle des ListViews übergeben.

4.2. Implementierung der grafischen Darstellung des Modells

In diesem Abschnitt geht es um die Implementierung der Visualisierung der verschiedenen grafischen Komponenten des Spiels. Dies beinhaltet die Darstellung des Raumschiffs, aber auch die Darstellung und Verwaltung des Hintergrunds, der zusätzlich angezeigten Sterne, sowie die Trajektorie der gewählten Funktion und der Trajektorie, welche durch den eigenen Verlauf erzeugt wird. Abschließend wird auf die Techniken zur Optimierung der Leistung der Grafikanzeige eingegangen.

[1] Toast: Eine Statusnachricht, die für kurze Zeit am unteren Bildschirmrand eingeblendet wird.

Für ein einfacheres Verständnis ist in diesem Abschnitt mit dem Begriff „Spiel“ der Teil des Apps gemeint, in dem das Raumschiff der Trajektorie folgen soll (sozusagen die Kernkomponente).

4.2.1. Zeitliche Diskretisierung der Aktualisierung der Daten des grafischen Elemente

Da die Aktualisierungsrate des Bildschirms nicht infinitesimal klein ist, ergibt sich auch keine Notwendigkeit, die Daten der Modelle ständig zu aktualisieren. Im Optimalfall werden die Daten der grafischen Elemente einmal pro Bildaufbau berechnet und dann angezeigt. Da dies jedoch nicht gewährleistet werden kann riskiert man mit dieser Vorgehensweise, dass die Daten beim Zeichnen noch nicht berechnet sind, und man daher zweimal das gleiche Bild zeichnet. Aus diesem Grund aktualisieren wir die Daten doppelt so oft, wie Bilder gezeichnet werden. Dies ist im Allgemeinen kein Problem, da Zeichnen bezüglich der Rechenzeit deutlich teurer ist als das Aktualisieren der Daten. Die maximal verfügbare Aktualisierungszeit T_a von 16.66 ms ergibt sich aus der Berechnung 4.1.

$$\begin{aligned} T_a &= c_{FPS}^{-1} \cdot \frac{1000ms}{2} \\ 16.\overline{66} &= 30^{-1} \cdot 500ms \end{aligned} \tag{4.1}$$

Die OpenGLrenderer-Klasse enthält daher einen Thread mit einer von außen unterbrechbaren Dauerschleife, der alle grafischen Elemente aktualisiert und danach überprüft wie viel Zeit der Aktualisierungsvorgang benötigt hat. Sollte der Vorgang vor der maximal verfügbaren Zeit beendet sein, so pausiert der Thread um die Zeit aufzufüllen und den Akkuverbrauch zu reduzieren. Sollte der Vorgang zu lang gebraucht haben wird sofort die nächste Aktualisierung durchgeführt.

4.2.2. Sterne

Zu Spielbeginn werden von der Renderer-Komponente 150 Stern-Objekte (im Folgenden „Stern“ genannt) angelegt. Jeder Stern ist ein zweidimensionaler Punkt in einem dreidimensionalen Raum. Beim Erzeugen des Sterns erzeugt dieser sich selbstständig eine zufällige Startposition auf dem Bildschirm und eine zufällige Startgeschwindigkeit. Die Geschwindigkeit liegt innerhalb fest vorgegebener Grenzen, damit deren spätere Positionsänderung aufgrund der Geschwindigkeit kein Flimmern verursacht. Damit die Darstellungsaspekte wie die Verteilung der Sterne über den Bildschirm möglichst gut aussehen haben die Sterne einige Eigenschaften, die ihre Darstellung beeinflussen:

Sichtbarkeit: Da sich der Stern linear auf der x-Achse fortbewegt tritt irgendwann der Fall ein, dass er nicht mehr sichtbar ist. Diese Information ist aus Leistungsgründen sehr wichtig. Wenn der Stern nicht mehr sichtbar ist (sich also außerhalb des Bildbereichs befindet) wird seine Position nicht mehr aktualisiert und er wird nicht mehr gezeichnet. Des Weiteren setzt er ein Flag, dass er erneut verwendbar ist. Somit ist es möglich, immer die gleichen 150 Sterne zu verwenden.

Geschwindigkeit: Jedes Mal, wenn der Stern erneut angezeigt wird, erzeugt er sich eine zufällige Geschwindigkeit (die eine obere und eine untere Schranke hat). Bei jedem Positionsupdate bewegt sich der Stern dann mit seiner Geschwindigkeit nach links (also entgegen der x-Achse).

Sternposition (Höhenwert): Um die Sterne auf der ganzen Höhe des Bildschirms anzuzeigen wird der Höhenwert mit einem Wert, der innerhalb der Bildschirmkoordinaten liegt, erzeugt. Der Höhenwert ändert sich jedes Mal, wenn der Stern neu angezeigt wird.

Sternposition (Breitenwert): Die x-Position des Sterns berechnet sich aus seiner Geschwindigkeit abzüglich der letzten Position. Wenn der Stern neu angezeigt wird, wird der Wert auf den rechten Rand des Bildschirms gesetzt.

Sternposition (Tiefenwert): Der Tiefenwert verursacht, dass die Sterne unterschiedlich groß dargestellt werden.

Erzeugungswahrscheinlichkeit: Um die Verteilung der Sterne zufälliger erscheinen zu lassen, werden diese wenn sie nicht mehr sichtbar sind nur mit einer konstanten Wahrscheinlichkeit zufällig zurückgesetzt. Diese Überprüfung findet bei jeder Positionsaktualisierung statt. Wir haben eine Wahrscheinlichkeit von 10% gewählt.

4.2.3. Hintergrund

Da die Sterne allein kaum Spielatmosphäre erzeugen, wird zusätzlich ein Hintergrund angezeigt. Dieser ist ein Bild, welches als Textur auf einer Ebene angezeigt wird. Der Hintergrund hat einen deutlich höheren Tiefenwert als die Sterne, damit er die Sterne nicht überdecken kann. Das Bild entstammt dem frei zugänglichen Bilderarchiv der NASA und ist frei verwendbar.

4.2.4. Dynamische Erstellung neuer grafischer Elemente

Da es nicht möglich ist, scheinbar endlose Objekte wie die Trajektorie der Strecke (siehe Abschnitt 4.2.5 auf der nächsten Seite) zu erzeugen, müssen diese dynamisch erzeugt werden. Aus diesem Grund haben die Sterne wie beschrieben einen eigenen Algorithmus um sich neu zu positionieren. Auch die Trajektorien von der vorgegebenen Funktion und dem Verlauf des Raumschiffes müssen nach und nach generiert werden. Bei der Trajektorie des Raumschiffes ist es beispielsweise nicht möglich die Trajektorie vorauszuberechnen und im Fall der Funktionstrajektorie würde dies sehr viel Speicher und Rechenzeit verbrauchen.

Aus diesem Grund haben wir einen Thread erzeugt, der die Verwaltung der Daten der Funktionstrajektorie übernimmt. Dieser führt jede Sekunde einen Aktualisierungsvorgang auf den Daten der Trajektorie aus. Die Funktionsweise des Algorithmus wird im folgenden Abschnitt erläutert.

4.2.5. Darstellung der Vorgabefunktion

Die Aktualisierung der Trajektorie der Strecke funktioniert weitestgehend unabhängig und ist deshalb nicht von der Synchronisierungsproblematik von Modell und Visualisierung des Raumschiffs betroffen.

Die Trajektorie ist in einzelne Segmente unterteilt. Jedes der Segmente hat einen Start- und einen Endpunkt, der linear interpoliert wird. Die Werte der Punkte wird während des Generierens des Segments aus der vorgegebenen Funktion entnommen. Es handelt sich also bei den Punkten und die tatsächlichen Funktionswerte, nicht um Annäherungen, die durch eine Interpolation entstanden wären.

Zu Beginn des Spiels wird eine bestimmte Anzahl an Segmenten der Trajektorie erzeugt und entsprechend gezeichnet. Wenn der soeben beschriebene Thread die Daten neu berechnet, wird in der Trajektorienklasse bestimmt, wie viele Elemente der Trajektorie nicht mehr sichtbar sind. Diese werden daraufhin entfernt und am vorderen Ende der Trajektorie werden entsprechend viele Segmenten wieder aufgefüllt. Die Trajektorie erstellt dabei einige Segmente als Puffer um den Fall eines verzögerten Aktualisierens durch den Thread kompensieren zu können.

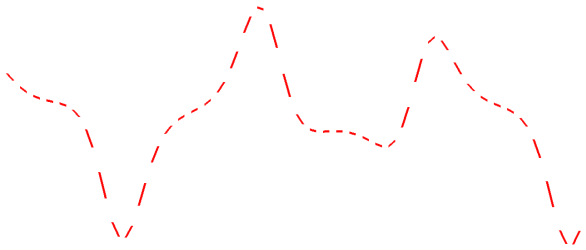


Abbildung 4.6.: Beispiel einer vom Spiel generierten Trajektorie.
Zur besseren Veranschaulichung wurde der Hintergrund entfernt.

4.2.6. Visualisierung des Verlaufs des Raumschiffs

Die Trajektorie des eigenen Raumschiffs wird als Punktfolge gezeichnet. Jedes Mal, wenn die Trajektorien erneut gezeichnet werden, wird überprüft, ob sich das Raumschiff seit dem letzten Erzeugen eines Punktes um einen fest vordefinierten Schwellwert „nach rechts“ bewegt hat. Wir haben uns für einen Schwellwert von 0,1 Einheiten entschieden, da es aus unserer Sicht ein gutes Verhältnis zwischen Information und Visual Clutter[2] ergibt.

4.2.7. Raumschiff

Die visuelle Darstellung des Raumschiffes ist als eine Fläche mit aufgetragener Textur realisiert. Die Textur hat dabei einen Alpha-Kanal, damit auch nur Teile der Textur gezeichnet werden, die auch zum Raumschiff gehören. Hierbei muss man beachten, dass

[2]Visual Clutter: Ein Zustand, bei dem die Fülle an Objekten oder deren Desorganisation zu einer Verringerung der Leistung einer bestimmten Aufgabenstellung führt [17].

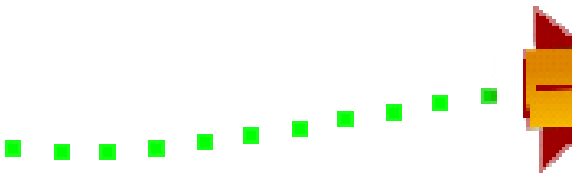


Abbildung 4.7.: Ausschnitt der vom Raumschiff ausgehenden Trajektorie

man unter OpenGL die Unterstützung von alpha-Kanälen, das so genannte *Alpha-Blending* explizit einschalten muss. Die Textureinbindung mit Alpha-Blending ist im Folgenden exemplarisch gegeben:

Listing 4.1: Exemplarischer Code zur Verwendung von Texturen mit Transparenz

```
gl.glEnable(GL10.GL_TEXTURE_2D);
gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
[...]
gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glDisable(GL10.GL_TEXTURE_2D);
```

Anmerkung: `textures[0]` ist die bereits zu einer Textur verarbeiteten Bilddatei.

Ein Raumschiff anzuzeigen, welches die eigene Orientierung nicht darstellt sorgt für Verwirrung beim Spieler. Aus diesem Grund wird beim Zeichnen unter Zuhilfenahme der aktuellen Geschwindigkeitswerte in x- und y-Richtung die Orientierung über die Formel 4.2 berechnet.

$$\alpha = -90^\circ + deg(\frac{v_y}{2 \cdot v_x}) \tag{4.2}$$

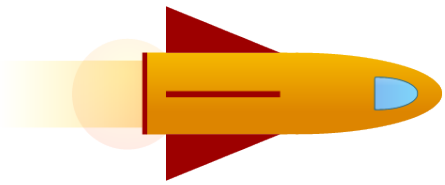


Abbildung 4.8.: Textur, die für das Raumschiff verwendet wurde.

Zusätzlich wird abgefangen, dass das Schiff nicht zu weit dreht und beispielsweise entgegen der Flugrichtung ausgerichtet ist.

4.2.8. Leistungsoptimierung

Im Folgenden werden die verschiedenen Techniken erläutert, die zur Leistungssteigerung des Spiels verwendet wurden.

Framerate-Limiter

In Abschnitt 4.2.1 auf Seite 32 wurde bereits erläutert, wie lange ein Zeichenvorgang Zeit hat, um seine Aufgabe zu beenden. Angenommen, diese Aufgabe wäre innerhalb der Zeit beendet, stellt sich die Frage, was in der noch verbleibenden Zeit geschehen soll. Ohne jegliche Optimierung wäre es naheliegend immer zu Zeichnen, wenn gerade Rechenzeit zur Verfügung steht. Dadurch ergeben sich jedoch einige Probleme. Zum einen würde ein Prioritätenmanagement benötigt, damit das Zeichnen nicht Rechenzeit von wichtigeren Aufgaben wie dem Aktualisieren der Datenmodelle wegnimmt und zum anderen wäre der Akkuverbrauch bei rechenstarken Endgeräten sehr hoch. Aus diesem Grund haben wir einen Framerate-Limiter in die `onDraw()`-Methode eingebaut. Dieser nimmt die Zeit zum Startzeitpunkt der Zeichenbefehle. Wenn alle Befehle abgearbeitet wurden wird erneut die Zeit genommen und deren zeitliche Differenz berechnet. Falls diese Differenz kleiner ist, als der Zeichenvorgang an Bearbeitungszeit zugesprochen bekommen hat, wird der Thread für die verbleibende Zeit pausiert. Auf diese Weise ist sowohl gewährleistet, dass der Akku geschont wird, als auch dass genügend Rechenzeit für die anderen Aufgaben zur Verfügung steht.

Face Culling

Zur Optimierung der Anzeige aller OpenGL-Oberflächen wurde in der OpenGL-Rendering-Klasse das so genannte *Face-Culling* eingeschaltet. Dies bewirkt, dass alle Oberflächen nur noch auf der Vorderseite gezeichnet werden (und die Rückseite nicht). Dies hat den Vorteil, dass man nur noch die Hälfte aller Oberflächen zeichnen muss. Dies reduziert die benötigte Rechenzeit und schont somit den Akku. Der Nachteil für den Entwickler ist, dass man darauf achten muss, in welcher Reihenfolge man die Eckpunkte der Formen (wie Triangle-Strip angibt, da diese bei falscher Handhabung falsch oder gar nicht angezeigt werden.

Dynamische Objektverwaltung

In Abschnitt 4.2.5 auf Seite 34 wurde bereits erläutert, dass zusätzliche Segmente dynamisch erstellt werden und gleichzeitig nicht mehr sichtbare Elemente gelöscht werden. Dies wurde implementiert, um ein Speicherleck im weiteren Spielverlauf zu verhindern.

Bei den Sternen ist die Vorgehensweise ähnlich. Da sich die Form der Sterne jedoch nicht ändert, können die nicht mehr sichtbaren Sterne einfach wiederverwendet werden. Dies spart Ressourcen, da das Erstellen und das Löschen von Objekten sehr teuer sind.

4.3. Schnittstelle zwischen Modell und Visualisierung

Beim Konzept des MVC kommt es noch mehr als in anderen Entwurfskonzepten auf die Schnittstellen zwischen Modell und Visualisierungen an. Deshalb werden im Folgenden die einzelnen Schnittstellen zwischen Modell und Visualisierung vorgestellt. Danach werden die konzeptionellen Probleme der Schnittstelle zwischen Visualisierung und Datenmodell angesprochen, woraufhin die Vor- und Nachteile der von uns gewählten Schnittstelle aufgezeigt werden und begründet wird, warum wir uns dennoch dafür entschieden haben.

4.3.1. Schnittstellen zwischen Modell und Visualisierung

Es müssen an verschiedenen Stellen Daten zwischen den verschiedenen Komponenten unseres Spiels übermittelt werden. Schon bevor das eigentliche Spiel beginnt müssen die vom Spieler gewählten Reglerparameter von der entsprechenden Activity an die Spiel-Activity übergeben werden. Hier muss der Entwickler noch keine eigene Schnittstelle entwickeln, da das Android Betriebssystem bereits ein probates Mittel vorgibt: Extras.

In den Grundlagen wurde bereits beschrieben, dass Activities durch Intents aufgerufen werden, welche dem Betriebssystem mitteilen, welche Activity gestartet werden soll. Intents werden jedoch nicht nur an das Betriebssystem übermittelt sondern können auch von der gestarteten Activity mittels `getIntent()` abgefragt werden. Dies ist sehr nützlich, da man in Intents nicht nur die zugehörige Activity-Klasse speichern kann, sondern auch noch weitere Informationen, die so genannten `Extras`. Diese Informationen sind vollkommen optional, doch können hiermit bestimmte vorgegebene Datentypen (hauptsächlich primitive Datentypen und teilweise Arrays von diesen) an einen Textschlüssel gebunden werden und dann mittels `getExtra(String key)` wieder abgerufen werden.

Auf diese Art werden bei uns die Parameter vor dem Spiel an die Spielactivity übermittelt. Mit dieser Technik wird außerdem nach dem Spiel die Punktzahl und die Informationen des Verlaufs an die Zusammenfassung übertragen.

Innerhalb des Spiels sind durch das MVC-Muster unser Datenmodell und die Visualisierung voneinander entkoppelt, doch hat die Visualisierung eine Referenz auf das Datenmodell um sich bei Bedarf deren Daten abzugreifen.

4.3.2. Konzeptionelle Probleme der Schnittstelle

Da Modell und Visualisierung voneinander unabhängig arbeiten ergibt sich eine Synchronisierungsproblematik. Hierfür gibt es keine perfekte Lösung, sondern nur den Versuch, die auftretenden Probleme so unauffällig wie möglich zu halten. Unsere Lösung war wie bereits angesprochen ein häufigeres Aktualisieren der visuellen Daten als das des Modells um eine scheinbar flüssige Wiedergabe der Daten zu ermöglichen. Dies geht jedoch mit einem erhöhten Ressourcenverbrauch einher, wodurch das Spiel möglicherweise auf leistungsschwächeren Smartphones die Animation zeitweise stoppen könnte.

4.3.3. Mögliche Lösungsansätze und deren Vor- und Nachteile

Um die konzeptionellen Probleme zu lösen bieten sich also drei Lösungsansätze an:

Kopplung von Visualisierung und Datenmodell: Eine Kopplung von Visualisierung und Datenmodell entspricht der Integration von dem Datenmodell in die Visualisierungsklasse und damit dem Gegenteil eines MVC. Hierdurch wäre ein mögliches Puffern von Werten zur schnelleren Abfrage unnötig und die Interaktion des Datenmodells mit dessen Visualisierung wäre deutlich einfacher. Dagegen spricht eine deutlich schlechtere Wartbarkeit und Erweiterbarkeit.

MVC mit einer 1:1 Synchronisation: Hierbei tritt das angesprochene Problem von möglichen Verzögerungen bei der Anzeige auf, was zu Bildsprüngen führen kann, was von den meisten Spielern als nicht flüssige Animation aufgefasst wird.

MVC mit Sampling Theorem: Analog zu Shannons Sampling Theorem [19], in welchem ein Bild als Funktion aufgefasst und dann zwecks einer Diskretisierung abgetastet wird, kann man das Datenmodell, in dem eine kontinuierliche Funktion ausgewertet wird, an der Schnittstelle entsprechend synchronisieren. Dabei muss man mindestens doppelt so oft die Daten der im Datenmodell hinterlegten Vorgabefunktion abgreifen, wie die Visualisierung aktualisiert wird. Dies ermöglicht ein perfektes Sampling der benötigten Daten.

Wir haben uns für den letzten Vorschlag entschieden. Zum einen können wir aufgrund des Sampling Theorems von einer perfekten Abtastrate ausgehen und zum anderen bietet das MVC viele Vorteile wie eine gute Wartbarkeit und Erweiterbarkeit. Zudem wäre es problemlos möglich, das Datenmodell oder die Visualisierung auszutauschen.

4.4. Highscore

Der Highscore ist ein weiterer wichtiger Aspekt eines jeden Spiels bei dem man Punkte erreichen kann. Dazu gehört natürlich die Punktanzeige, aber auch die Berechnung der Punkte, wie die ermittelten Punkte an den Highscore übergeben werden und die Implementierung des Datenmodells des Highscores.

Berechnung der Punkte

Wenn das Spiel gestartet wird, beginnt im Hintergrund die Berechnung des Highscores. Dabei wird ein Thread erzeugt, der nebenläufig alle 250 Millisekunden die Differenz zwischen der vorgegebenen Trajektorie und der Position des Schiffes berechnet. Wenn diese identisch sind werden 10 Punkte vergeben. Mit zunehmendem Abstand verringert sich die vergebene Punktzahl s linear. Die Berechnungsformel entspricht dabei den Formeln aus 4.3.

$$\begin{aligned} s &= ||\lfloor 10 \cdot (y_{Position} - y_{Vorgabe}) \rfloor|| \\ y_{Vorgabe} &= f_{Vorgabefunktion}(x_{aktuell}) \end{aligned} \tag{4.3}$$

Es sei in diesem Zusammenhang darauf hingewiesen, dass die Differenz mit einer Präzision von 64 Bit berechnet wird, was die Vergabe von 10 Punkten praktisch ausschließt.

Die berechneten Punkte werden direkt nach deren Ermittlung aufsummiert. Wenn das Spiel endet, ruft eine Methode die Punktzahl ab und schreibt diese für die Verwendung in der Zusammenfassung in die Extra-Informationen der Activity. Außerdem wird noch vor dem Beenden des Spiels die Punktzahl in die Datenbank geschrieben.

4.4.1. Verwaltung des Highscores

Der Highscore wird in einer SQLite-Datenbank verwaltet. SQLite bietet sich an, da das Android OS diese Art von Datenbank nativ unterstützt und der Entwickler keinen zusätzlichen Aufwand hat. Die reduzierte Funktionalität von SQLite kann in unserem Fall vernachlässigt werden, da unsere Datenbank ohnehin nur aus einer Tabelle besteht. In diese Tabelle können im Prinzip alle Daten des Programms geschrieben werden, das Spiel schreibt jedoch nur den Namen, die Punktzahl, den Modus und die vorgegebene Funktion in die Datenbank, da nur diese Werte für die Anzeige benötigt werden.

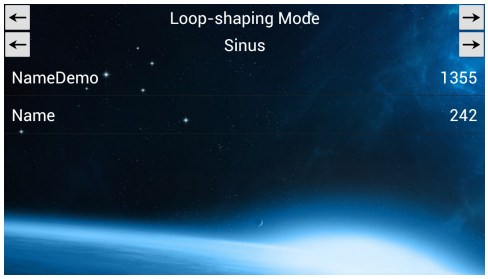


Abbildung 4.9.: Screenshot unserer Implementierung der Ansicht des Highscores

4.4.2. Visualisierung des Highscores

Die Android-API stellt nativ eine Listenansicht zur Verfügung. Um diese zu nutzen, muss man statt einer normalen Activity eine sog. 'ListActivity' verwenden. Ein Highscoreeintrag soll dabei den Namen des Spielers und die erreichte Punktzahl enthalten.

Standardmäßig unterstützt Android zwar die Anzeige von zweiwertigen Tabelleneinträgen, doch werden diese Informationen dann untereinander angezeigt. Da wir jedoch ein Layout anstrebten, das die Breite des Displays besser ausnutzt haben wir uns entschieden selbst ein Layout zu entwerfen, welches die beiden Werte nebeneinander anzeigt. Dieses wurde als eine Art Template[3] übergeben. Die anzuzeigenden Werte werden über eine SQL-Datenbankabfrage in einem Zeiger gespeichert, der an die Android-Klasse `SimpleCursorAdapter` übergeben wird. Dieser wird des Weiteren das Layout und die anzuzeigenden Felder übergeben. Diese Klasse kann dann an die Methode `setListAdapter(SimpleCursoradapter dataSource);` übergeben werden, woraufhin das System die Listenansicht automatisch erzeugt.

[3]Template(engl.) ≙ Schablone(dt.): Ein von außen vorgegebenes Layout, welches dynamisch eingefügt werden kann.

5. Zusammenfassung und weiterer Ausblick

In dieser Arbeit wurden nach einer kurzen Einleitung und den Grundlagen der Regelungstechnik und des Betriebssystems Android (siehe Kapitel 2) die Architektur des Spiels und die dabei benötigten Komponenten vorgestellt. Hierbei wurde auf die genutzten Konzepte und Entwurfsmuster der Softwaretechnik eingegangen. Zudem wurden auch die Komponenten kurz vorgestellt, die nicht der Hauptfokus dieser Arbeit waren, wie beispielsweise Menüführung und Sensorauswertung (siehe Kapitel 3). Anschließend wurde im Implementierungskapitel (siehe Kapitel 4) zunächst der Ablauf des Spiels skizziert. Daraufhin wurde vor allem auf die konkrete Implementierung der Visualisierungskomponente im Detail eingegangen. Daraufhin wurden die notwendigen Schnittstellen zwischen Regler- und Visualisierungskomponente, sowie deren konzeptionelle Probleme und Lösungsvorschläge vorgestellt. Zuletzt wurde auf die Verwaltung und Visualisierung des Highscores eingegangen.

Es wurden zwar alle Forderungen der Aufgabenstellung erfüllt, doch wäre es denkbar, das Spiel noch zu erweitern. So wäre es vorstellbar, die Grafik entsprechend der Leistungsfähigkeit des Endgerätes anzupassen. Zudem ist eine Erweiterung des Spiels um weitere Spielmodi nicht ausgeschlossen. Die Spielzusammenfassung könnte um mehr Interaktionsmöglichkeiten wie ein Zoomen in der Verlaufsübersicht ergänzt werden und der Highscore könnte kontextabhängig die Punktzahl des Spielers anzeigen und hervorheben.

A. Grundlagen der Regelungstechnik

Im Folgenden werden die im Kontext dieser Studienarbeit wichtigsten Aspekte der Regelungstechnik vorgestellt. Für nähere Informationen sei auf [10] verwiesen. Teile der Abbildungen entstammen der Vorlesung „Einführung in die Regelungstechnik“ von Prof. Dr.-Ing. Christian Ebenbauer [15].

A.1. Modellgleichung / Modellklasse

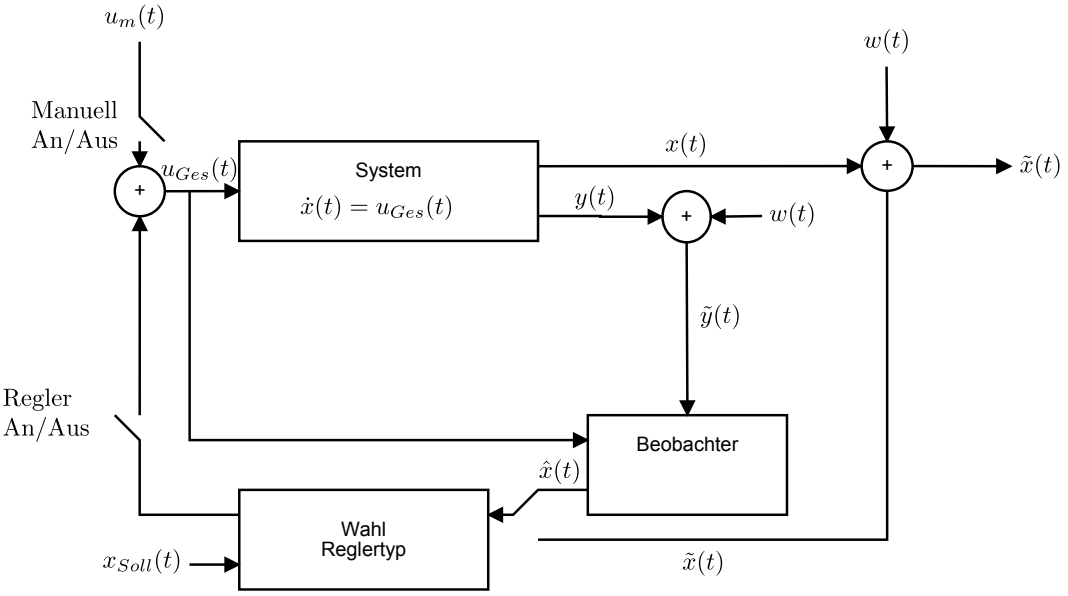


Abbildung A.1.: Systemmodell der Reglerkomponente

Zustandsrückführung

$$u = -k^T x + vr \tag{A.1}$$

Luenberger Beobachter

$$\dot{\hat{x}} = A\hat{x} + bu + l(y - \hat{y}) \tag{A.2}$$

$$u = -k^T x + vr \tag{A.3}$$

Loop-Shaping

$$K(s) = kv\frac{(s + w)}{(s + vw)} \tag{A.4}$$

A.2. Pol-Diagramm

Die Darstellung der Pole wird mit Hilfe des Pol-Diagramms vorgenommen. Anhand dieses Diagramms können Aussagen über die Stabilität eines Systems gemacht werden. Befinden sich alle Pole einer Übertragungsfunktion in der linken offenen Halbebene des Diagramms, so ist das System stabil.

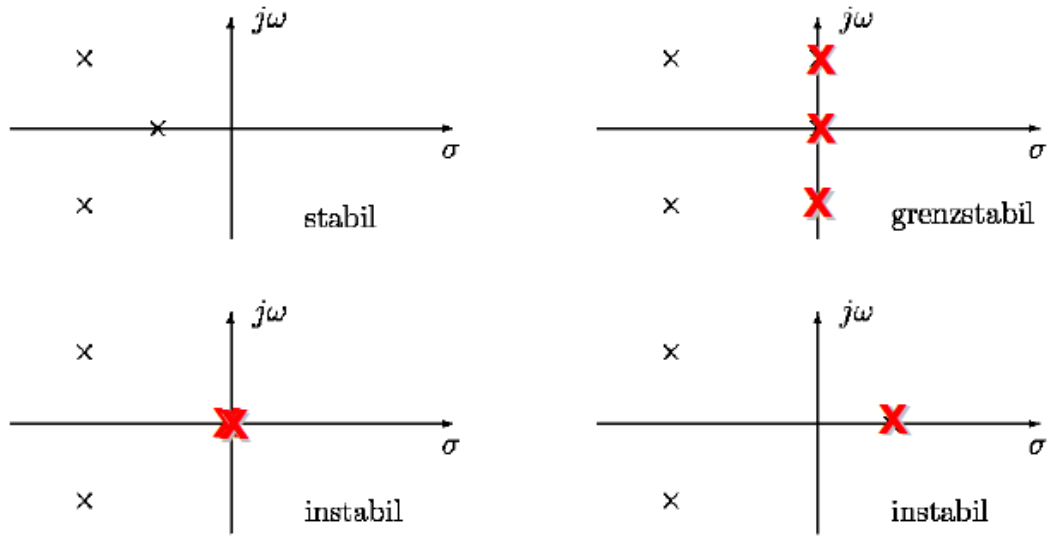
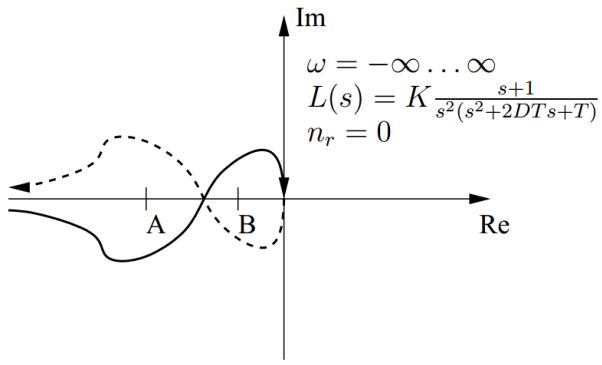


Abbildung A.2.: Illustration verschiedener Pol-Nullstellen-Diagramme

A.3. Nyquistdiagramm

Als Nyquistdiagramm wird in der Regelungstechnik die Ortskurve des Frequenzgangs der Übertragungsfunktion bezeichnet. Dabei wird lediglich die Übertragungsfunktion des offenen Regelkreises $G_0(s)$ betrachtet um dadurch Rückschlüsse auf die Stabilität des geschlossenen Kreises zu ziehen. Gezeichnet wird das Diagramm, indem man sämtliche möglichen Werte (0 bis ∞) in den Funktionsparameter der Übertragungsfunktion $G_0(j\omega)$ einsetzt.



$A = -1 + 0j$, $\Delta\arg(1 + L(j\omega)) = \pi = \frac{\pi}{2}(2 + 2 \cdot 0) \Rightarrow$
$T(s) \in$BIBO .
$B = -1 + 0j$, $\Delta\arg(1 + L(j\omega)) = -\pi \neq \frac{\pi}{2}(2 + 2 \cdot 0) \Rightarrow$
$T(s) \notin$BIBO.

Abbildung A.3.: Illustration eines Nyquistdiagramms

Mit Hilfe des Nyquistdiagramms hat der schwedisch-amerikanische Physiker Harry Nyquist folgendes Stabilitätkriterium für den (geschlossenen) Regelkreis formuliert.

A.3.1. Satz (Nyquist-Stabilitatskriterium)

Sei $\Delta\phi\{1 + G_0(j\omega)\}$ die Phasendrehung des offenen Kreises $G_0(j\omega)$ bzgl. des Punktes (-1|0) wenn ω den Wertebereich von 0 bis ∞ durchläuft. Sei m_0 die Anzahl der Pole von G_0 in der rechten Halbebene (d.h. mit positivem Realteil) und a_0 die Anzahl der Pole auf der imaginären Achse (d.h. mit Realteil Null). Dann gilt: Der geschlossene Kreis ist genau dann asymptotisch stabil, wenn

$$\Delta\phi\{1 + G_0(j\omega)\} = m_0\pi + a_0\frac{\pi}{2}. \tag{A.5}$$

A.4. Bodediagramm

Die getrennte Darstellung von Amplitudengang und Phasengang des Frequenzganges G(jω) über die Frequenz ω wird Frequenzkennliniendiagramm oder Bode-Diagramm genannt.

Dabei werden Amplitude und Phase in zwei getrennt Schaubilder gezeichnet.

$$\begin{aligned} Amplitude : A(\omega) &= |G(j\omega)| \\ Phase : \Delta\phi &= arc(G(j\omega)) \end{aligned} \tag{A.6}$$

Die Frequenz ω wird auf der Horizontalachse in rad/s angegeben. Die Frequenzachse wird mit einer logarithmischen Skalierung aufgetragen. Darüber hinaus wird der Betrag

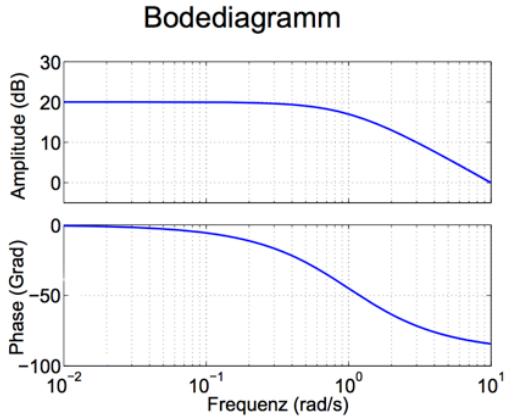
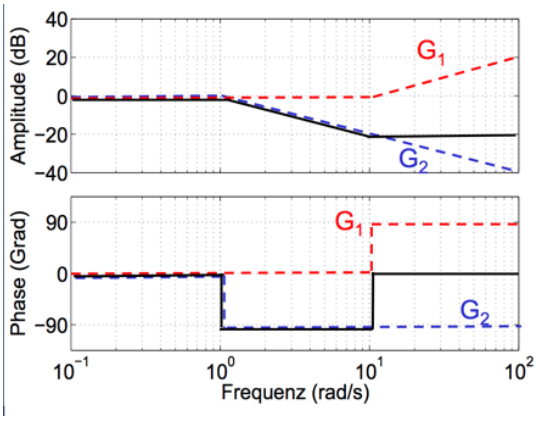


Abbildung A.4.: Illustration eines Bodediagramms

des Amplitudengangs |G(jω)| im Bode-Diagramm als $20 \cdot log|G(j\omega)|$ in Dezibel (db) aufgetragen. Die Phase wird in Grad oder rad aufgetragen. Die Phase wird im Gegensatz zu den anderen Achsen nicht logarithmisch dargestellt. Die logarithmische Darstellung bietet einige Vorteile, so kann beispielsweise eine einfache Approximation mit Geraden vorgenommen werden. Ein weiterer großer Vorteil ist, dass der Amplitudengang und der Phasengang zweier in Reihe geschalteter Systeme durch Addition der einzelnen Amplituden- bzw. Phasengänge gefunden werden kann.



Amplitudengang

$$G(j\omega) = G_1(j\omega)G_2(j\omega) \tag{A.7}$$

$$A(\omega) = |G(j\omega)| = |G_1(j\omega)||G_2(j\omega)| = A_1(\omega)A_2(\omega) \tag{A.8}$$

$$20log(A(\omega)) = 20log(A(\omega)A(\omega)) = 20log(A(\omega)) + 20log(A(\omega)) \tag{A.9}$$

Phasengang

$$arc(G(j\omega)) = arc(G_1(j\omega)G_2(j\omega)) = arc(G_1(j\omega)) + arc(G_2(j\omega)) \tag{A.10}$$

B. Systemarchitektur des Spiels

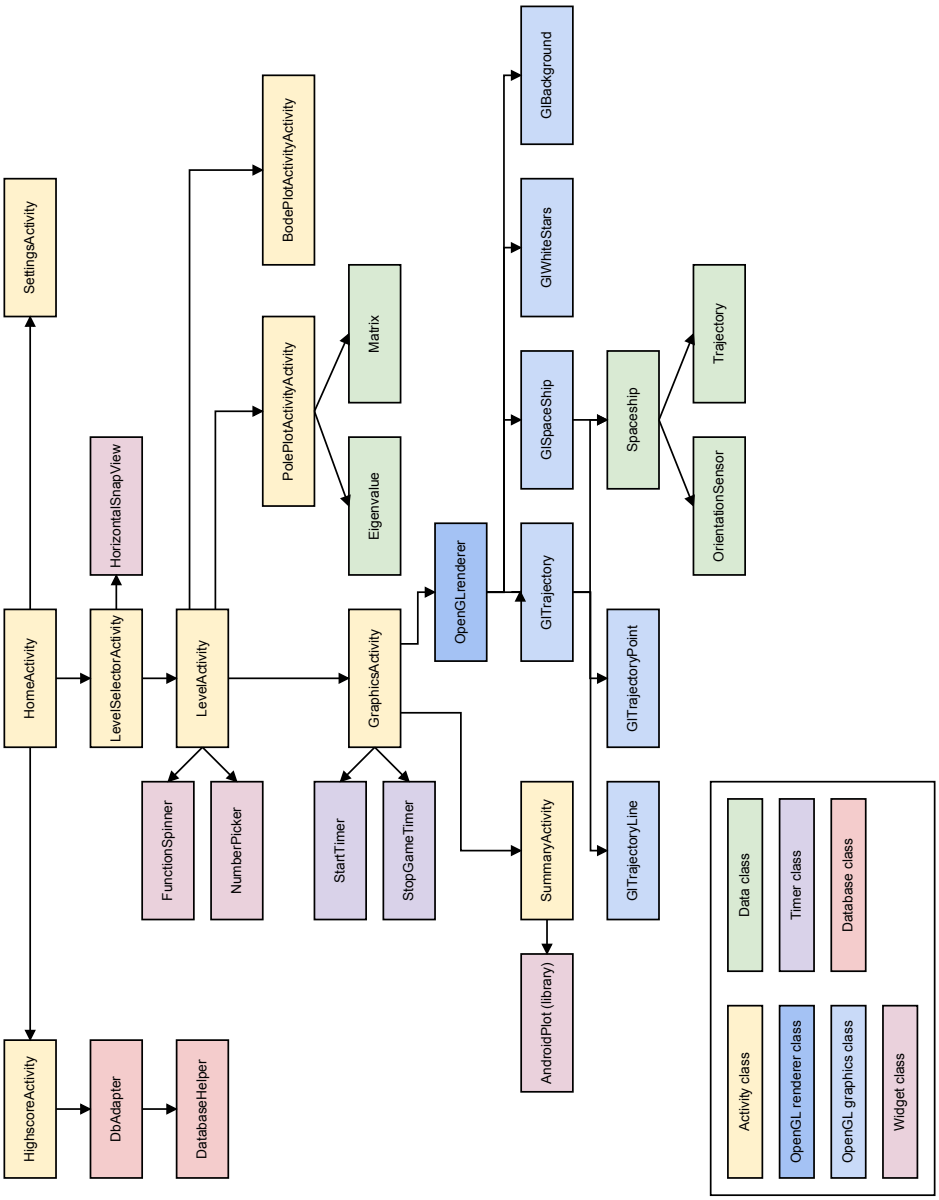


Abbildung B.1.: Vollständige Systemarchitektur des Spiels

Literaturverzeichnis

[1] Gnome 3 website. http://www.gnome.org/gnome-3/.

[2] Focus Online. Samsung baut Spitzenstellung in Deutschland aus, 02 2013. http://www.focus.de/digital/internet/netzoekonomie-blog/smartphones-samsung-baut-spitzenstellung-in-deutschland-aus_aid_916120.html.

[3] Google Inc. Activities. http://developer.android.com/guide/components/activities.html.

[4] Google Inc. Android lifecycle. http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle.

[5] Google Inc. Android NDK.

[6] Google Inc. Installing the SDK. http://developer.android.com/sdk/installing.html.

[7] Google Inc. Source of Android drivers. http://developer.android.com/sdk/oem-usb.html.

[8] Google Inc. What is Android? http://developer.android.com/guide/basics/what-is-android.html.

[9] Heise Verlag. Jeder dritte Deutsche hat ein Smartphone. http://heise.de/-1526048.

[10] Horn, M., and Dourdoumas, N. *Regelungstechnik-Bafög-Ausgabe: Rechnerunterstützter Entwurf zeitkontinuierlicher und zeitdiskreter Regelkreise*. Addison Wesley in Pearson Education Deutschland, 2006.

[11] IDC. Strong Demand for Smartphones and Heated Vendor Competition Characterize the Worldwide Mobile Phone Market at the End of 2012. http://www.idc.com/getdoc.jsp?containerId=prUS23916413#.UROvWmdI3IV.

[12] Khronos Group. OpenGL ES - The Standard for Embedded Accelerated 3D Graphics. http://www.khronos.org/opengles/.

[13] Merg, M. Entwicklung eines Regelungstechnik Spiels für Android Smartphones - Modell und Regelung, 2012.

[14] Mono Project. Cross platform, open source .NET development framework. http://www.mono-project.com/.

[15] Prof. Dr.-Ing. Christian Ebenbauer. Einführung in die Regelungstechnik, 2010/11.

[16] Rideout, P. *iPhone 3D Programming: Developing Graphical Applications with OpenGL ES*. O'Reilly Media, 2010.

[17] Rosenholtz, R., Li, Y., Mansfield, J., and Jin, Z. Feature congestion: a measure of display clutter. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2005), ACM, pp. 761–770.

[18] Schmiedl, G. Mobile enabling of virtual teams in school: an observational study on smart phone application in secondary education. In *Education Technology and Computer (ICETC), 2010 2nd International Conference on*.

[19] Unser, M. Sampling-50 years after shannon. *Proceedings of the IEEE 88*, 4 (2000), 569–587.

[20] wikipedia.org. Definition SDK. http://de.wikipedia.org/wiki/Software_Development_Kit.

[21] wikipedia.org. Head-up-Display. http://de.wikipedia.org/wiki/Head-up-Display.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 25.4.2013