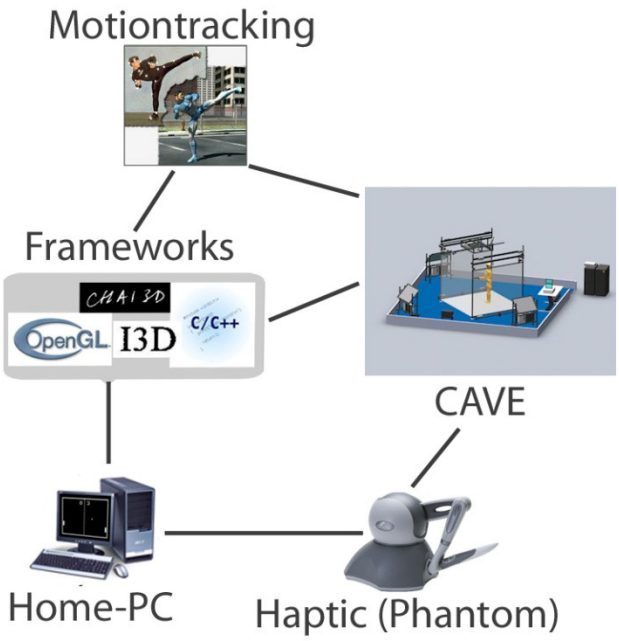
Bachelor Thesis 2012

SquashI3D

Dokumentation – V1.00



Fachbereich	BFH-TI Informatik
Studierende	Andreas Emch
	Daniel Pfäffli
Professoren	Prof. Urs Künzler

Management Summery

Aufgabe

SquashI3D bezeichnet ein Computerspiel, welches für den Einsatz im CAVE - Computer Automated Virtual Environment - entwickelt wurde. Das Spiel ist eine Virtualisierung und Adaption der Sportart Squash. Dabei kommen haptische Geräte zum Einsatz, welche das genaue Steuern eines virtuellen Objektes im Raum zulassen. Das eingesetzte Gerät Phantom Sensable Omni bietet 6-Freiheitsgrade an, die alle verwendet werden.

Situation

Der CAVE ist ein Raum der Berner Fachhochschule, bestehend aus vier Leinwänden und ermöglicht auf diese Weise das Begehen einer virtuellen Welt. Die Umsetzung für Applikationen im CAVE wird mit dem haus-eigenen Framework I3D gemacht. Dies ist ebenfalls die Basis für die Applikation SquashI3D. Das I3D-Framework vereinigt verschiedene bekannte Frameworks und stimmt sie für den Einsatz im CAVE ab. Nachfolgend werden die wichtigsten Frameworks erklärt.

Das Equalizer-Framework ermöglicht das Betreiben einer Applikation über mehrere Computer hinweg. Dies ist zwingend, da die Berechnung der Szene pro Leinwand auf unterschiedlichen Computern geschieht. Der Einsatz dieses Frameworks erhöht die Komplexität der Software-Architektur, weil auf die Synchronhaltung der einzelnen Applikationsinstanzen Rücksicht genommen werden muss.

Mit dem Framework Bullet wird die Physik abgedeckt. Die Integration im I3D-Framework ist eng mit dem Framework Chai3D verbunden. Chai3D bezeichnet ein Framework speziell für haptische Geräte. Das I3D-Framework war bisher auf maximal ein haptisches Gerät ausgelegt.

Für die Visualisierung dient das Framework OSG. I3D stellt eine gute Verknüpfung von OSG und der Physik Bullet zur Verfügung.

Strategie

Unser Vorgehen war angelehnt an das Prototyping-Verfahren. Für die Analyse Phase haben wir mit Hilfe verschiedener Prototypen den IST-Zustand ermittelt. Die wichtigsten Erkenntnisse aus dieser Phase sind:

- Die Anforderung INCA 6D (anderes haptisches Gerät) ist für uns nur mit einem enormen Mehraufwand abzudecken.
- Die Anforderung 2-Haptic Geräte braucht mehr Analysezeit, ist aber umsetzbar.

Die Abdeckung einzelner Anforderungen geschah mit Hilfe von Prototypen. Diese Prototypen wurden ständig um die neue Funktionalität erweitert. In regelmässigen Abständen testeten wir unsere Applikation im CAVE.
Diese Strategie erforderte, dass die Synchronisation der Applikationsinstanzen zu den ersten Anforderungen gehörte, die wir umgesetzt haben. Diesen Bereich haben wir EventSystem getauft.

Umsetzung

Die Synchronisation der einzelnen Applikationsinstanzen kann wie folgt dargestellt werden.

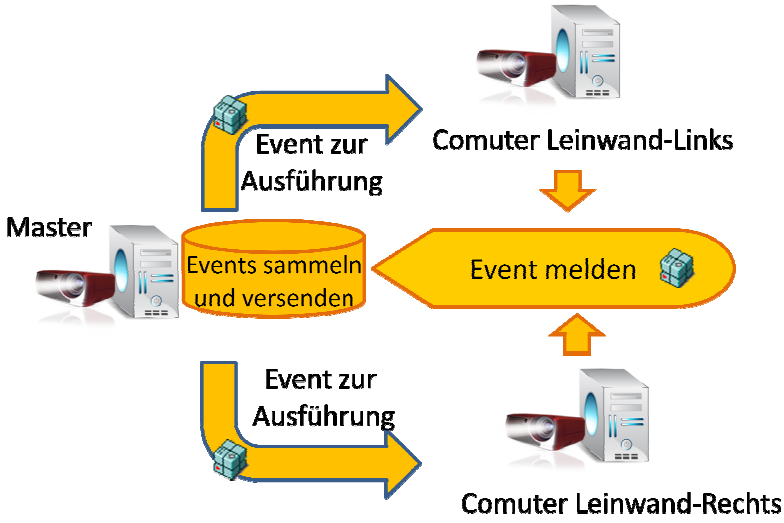


Abbildung 1: Schematische Darstellung EventSystem

Als Basis Einheit für die Synchronisation werden Events versendet. Diese müssen zum Master gesendet werden, welcher die Events sammelt und an alle Applikationsinstanzen weiterleitet. Die wichtigsten Events aufgezählt:

- Bewegungen von Ball und Spieler-Racket
- Spielzustand
- Benutzeraktionen

Der Benutzer kann zwischen Game-Levels zur Laufzeit wechseln. Das heisst, es stehen mehrere Squash-Räume zur Verfügung. Diese Anforderung haben wir mit Hilfe einer dynamischen Szenenerstellung realisiert. Das Format ist als XML aufgebaut.

Abbildung 2: Szenenwechsel im Menu - XML-Aufbau

Um die Spielregeln abbilden zu können, haben wir eine GamePlay-StateMachine entwickelt. Diese BallStateMachine liest aus den Kollisionen der Physik-Engine die Geschehnisse und wandelt diese in einen Spielzustand um. Beispiel: Ball trifft auf Racket von Spieler 1, also muss der Ball als nächstes an die Wand.

Die Anzahl haptischer Geräte ist in SquashI3D nur durch die Logik und das Framework für Haptic Chai3D beschränkt. Dafür mussten wir die haptische Integration von I3D zu einem grossen Teil umschreiben. Die Umsetzung hebt die Einschränkung von I3D, dass ein haptisches Gerät nur eine Kugel steuern kann, auf. Es können beliebige Objekte gesteuert werden.

Fazit

Das Spiel SquashI3D erweitert das I3D-Framework, um die Möglichkeit beliebig viele haptische Geräte verwalten und Events (bisher waren nur statische Werte möglich) versenden zu können. Die Applikation besteht aus verschiedenen Szenen, die sich durch Spielräume unterscheiden, sowie verschiedene Menu-Szenen, welche zur Laufzeit ausgetauscht werden. Die Szenen sind dynamisch als XML-Format definiert.
Wir haben die Anforderungen Priorität 1 zu 100% erfüllt. Ausnahme bildet das haptische Feedback, welches wegen möglicher Beschädigung der Geräte abgeschaltet wurde. Die problematische Berechnung der haptischen Kräfte war nicht als Teil der Arbeit geplant. Wir haben viele Priorität 2 und ebenfalls wenige Priorität 3 Ziele umgesetzt.
Insgesamt ist es eine gelungene Arbeit und durchaus als Beta-Version eines Computerspiels vertretbar.
Dies ist das erste Spiel, für welches die Haptic im CAVE eingesetzt werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Allgemeines

1.1 Ziel und Zweck des Dokumentes

Dieses Dokument beschreibt das Projekt SquashI3D und die geleisteten Arbeiten.

1.2 Projektteam

Rolle / Rollen	Name	E-Mail
Dozent	Urs Künzler	urs.kuenzler@bfh.ch
Experte	Andreas Dürsteler	
Student	Andreas Emch	emcha1@bfh.ch
Student	Daniel Pfäffli	pfafd1@bfh.ch

Tabelle 1: Projektteam

1.3 Versionierung

Version	Datum	Bemerkung
0.10	30.11.2012	Erst-Version der Dokumentation. Prüfen, durch Herrn Künzler, ob die Struktur stimmt.
0.50	31.12.2012	Überarbeitete Version, hinzufügen von den verschiedenen Themen, die in unserer Arbeit aufgetaucht sind.
0.80	03.01.2012	Weitere Themengebiete beschreiben
0.90	09.01.2012	Weitere Themengebiete beschreiben
1.00	17.01.2012	Finalisieren und Drucken

Tabelle 2: Versionierung

1.4 Inhalte der DVD

Verzeichnis/ Datei	Beschreibung
/documentation	Enthält alle Dokumente.
/D_BT_HapticGame_V1.00.pdf	Dokumentation Bachelor-Thesis
/PA_BT_HapticGame_V1.10.pdf	Pflichtenheft
/appendix	Zusätzliche Dokumente
/J_BT_HapticGame_V1.00.pdf	Arbeitsjournal von Andreas Emch und Daniel Pfäffli
/S_BT_HapticGame_V1.00.pdf	Sitzungsprotokoll
/drivers	Verzeichnis, welches die notwendigen Treiber enthält
Phantom_Device_Drivers_ 5.1.7_Release.exe	Treiber für SensAble Phantom Omni
/sources	Enthält die Source-Dateien
/DoxyGen	Verzeichnis für die Source-Code Dokumentation.
/index.html	Start-Datei für die Source-Code Dokumentation.
/SquashI3D-Trunk	Root-Ordner für Source-Dateien. Enthält cmake-Dateien.
/SquashI3D-64Bit	Beinhaltet eine für 64-Bit Systeme kompilierte Version von SquashI3D.
/SquashI3D.exe	Start-Datei für die Applikation SquashI3D.

Tabelle 3: Inhalte der DVD

2 Projektbeschreibung

Die Aufgabe dieser Bachelor-Thesis ist es, ein Spiel umzusetzen, welches im CAVE eingesetzt werden kann. Für die Steuerung soll dabei auf die Haptik oder MotionTracking zurückgegriffen werden. Dies bedingt den Einsatz des Frameworks I3D, welches von der Berner Fachhochschule entwickelt und gewartet wird. Das I3D Framework setzt Equalizer ein, ein Framework für paralleles Rendering, um die Applikation auf vier Leinwänden projizieren zu können.
Als Ausgangslage wurde im Rahmen des Projektes 2 bereits eine Applikation, basierend auf dem I3D-Framework, umgesetzt. In dieser Arbeit haben wir uns hauptsächlich mit den in I3D integrierten Frameworks Chai3D (Haptik), OSG (Visualisierung) vertraut gemacht. Für die Bachelor-Arbeit muss für die Applikation weitere Frameworks, wie Equalizer, Bullet (Physik-Engine), beachtet werden.
Das angestrebte Spiel soll die Vorteile des CAVE nutzen und eine 3D-Welt darstellen können. Mit Hilfe der Eingabegeräte soll der Benutzer den Raum erleben können.

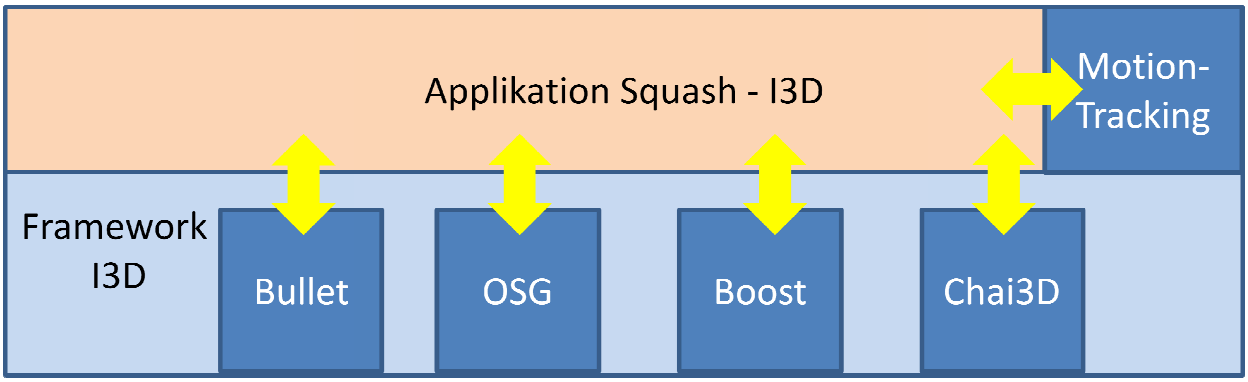


Abbildung 3: Framework – Übersicht

3 Funktionsweise Equalizer

Das von der Berner Fachhochschule entwickelte Framework I3D integriert unterschiedliche Frameworks, um die Anforderungen des CAVE erfüllen zu können. Das Equalizer-Framework ist ein System, welches erlaubt, eine Applikation gleichzeitig auf mehreren Computern laufen zu lassen. Das Framework bietet Hilfsmittel um diese Applikationsinstanzen, wie eine einzige Applikation laufen zu lassen. Normale Grafikkarten haben bis zu 5 Monitor Ausgänge. Mit diesem Framework können jedoch x-beliebige Computer zusammengeschlossen werden. Damit stehen x-beliebig viele Monitorausgänge zur Verfügung, über welche die Applikation angezeigt werden kann.

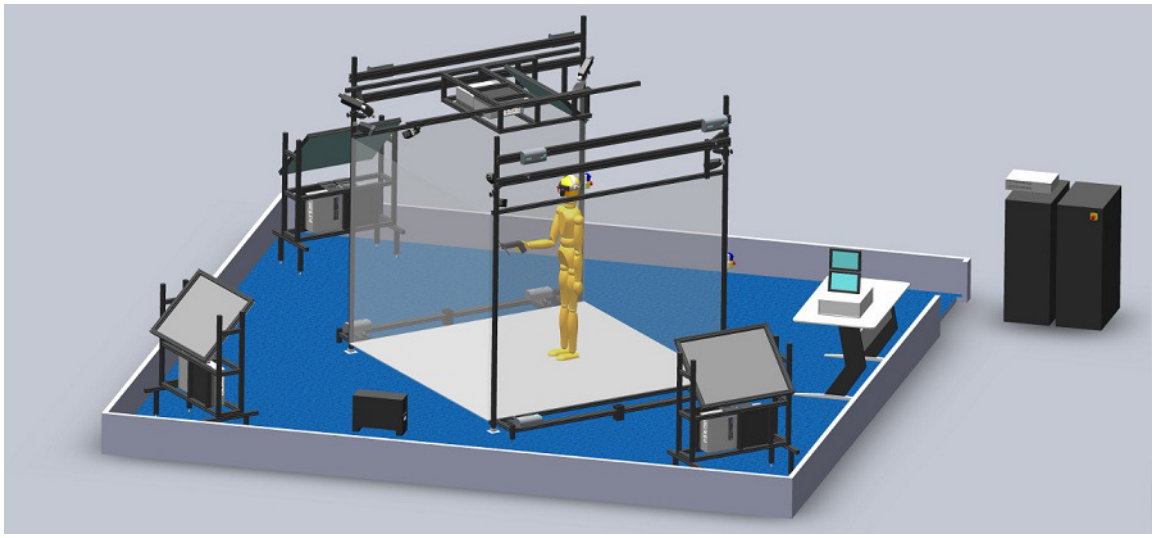


Abbildung 4: Darstellung CAVE

Genau diesen Vorteil setzt das Framework I3D ein, um die verschiedenen Beamer (insgesamt 8) ansteuern zu können.
Der Einsatz eines solchen Frameworks erhöht die Komplexität einer Applikation. Als Konsequenz müssen folgende Fragestellungen geklärt werden:

- Wie werden Änderungen am System an die anderen Computer übermittelt/ synchronisiert? (z.B. Tastatureingaben)
- Was passiert, wenn an zwei Computer denselben Befehl starten? Welcher gewinnt?

Da das Framework Equalizer sich nicht auf Computer, sondern vielmehr auf GPU-Ausgänge konzentriert, stellen sich weitere Fragen. Nehmen wir an die Applikation läuft auf einem Computer mit zwei GPU-Ausgängen.

- Wird die visuelle Szene einmalig auf dem System gehalten oder pro GPU-Ausgang?
    - Falls pro GPU-Ausgang, muss jede Aktion (z.B. steuern eines Elementes mit der Tastatur) auf beiden Szenen durchgeführt werden
    - Falls einmalig, muss der Zugriff kontrolliert werden, da die beiden GPU-Ausgänge gleichzeitig darauf zugreifen wollen.

3.1 Equalizer im I3D

Die Synchronisation geschieht im I3D über ein Objekt einer Klasse namens FrameData. In der folgenden Graphik ist dies sehr vereinfacht dargestellt.

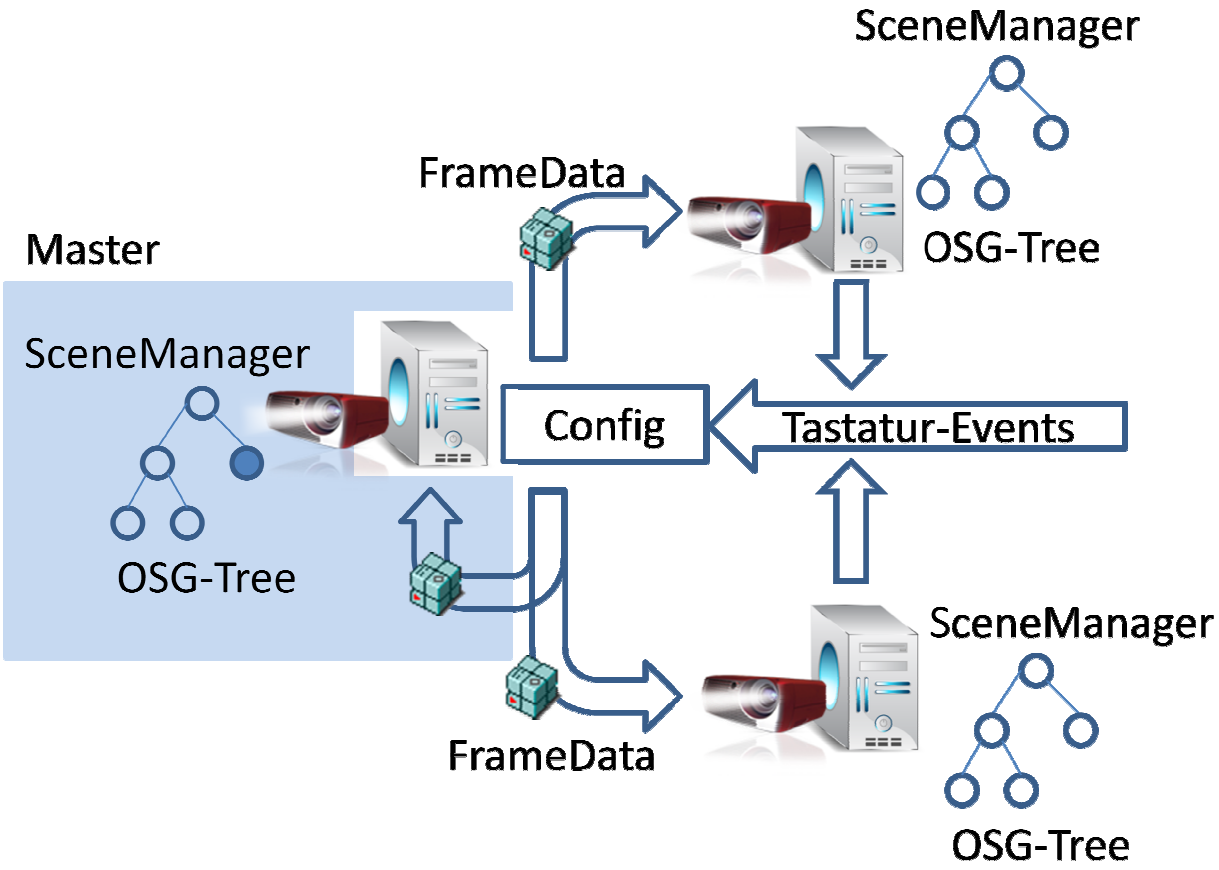


Abbildung 5: Synchronisation der Applikation

Das FrameData enthält alle Veränderungsanfragen am OSG-Tree. Diese werden auf die anderen Equalizer-Computer (Nodes) synchronisiert und auf den OSG-Tree angewandt. Unter Veränderungsanfragen wird verstanden, wie oft z.B. die Pfeil-Taste in einer Periode gedrückt wurde. Im Gegenzug werden Tastatur-Events über die Config-Klasse an den Master gesendet. Dieser nimmt die Änderungen entgegen und verpackt sie in einem neuen FrameData, das anschliessend versendet wird.
I3D synchronisiert ausschliesslich die Position des Betrachters. Dies bedeutet, aus welchem Blickwinkel der Betrachter die Szene sieht. Alles weitere muss die jeweilige Applikation selber lösen.

Im I3D wird der OSG-Tree auf dem SceneManager verwaltet. Diese Klasse gibt es pro Applikation (Node) einmal. Das bedeutet, dass der OSG-Tree nur einmal verwaltet wird, jedoch muss der Zugriff kontrolliert werden (damit nicht, gleichzeitig auf Ressourcen zugegriffen werden kann).

Werden an zwei Computer zwei Befehle gestartet, so werden diese nacheinander ausgeführt. Der letzte gewinnt.

3.2 Einschränkungen für SquashI3D

Die Haptik-Events müssen über das FrameData synchronisiert werden. Dies gilt ebenfalls für die Applikationszustände (Einstellungen, Spielstand, Aktionen).
Die Zugriffe auf den SceneManager, sowie andere Ressourcen, welche nicht pro GPU-Ausgang (Pipe) verwaltet werden, müssen unter Umständen vor doppelten Zugriffen geschützt werden. Damit kann gegenseitiges überschreiben von Daten und daraus resultierende möglichen Applikationsfehlern vorgebeugt werden.

4 Systemdokumentation

In diesem Abschnitt wird das SquashI3D als Applikation erklärt.

4.1 Ziele

Die Ziele wurden im Dokument PA_BT_HapticGame_V1.10.pdf, Kapitel 6.11 Priorisierung erläutert. Wir verzichten darauf die Tabelle hier ebenfalls anzugeben.
Der Erfüllungsgrad der Ziele ist in Kapitel 10.1 Erfüllungsgrad der Anforderungen beschrieben.

4.2 Hardware

Unsere Entwicklungsumgebung unterscheidet sich in folgenden entscheidenden Komponenten vom CAVE:

	CAVE	**Entwicklungsumgebung**
Prozessor-Typ	Intel	AMD
Grafikkarten-Type	Nvidia	AMD

Tabelle 4: Hardware Differenzen

Wir haben festgestellt, dass das Framework Equalizer auf spezifische Features von Nvidia zugreift. Dies bedeutet für AMD Grafikkarten, dass die gleichen Berechnungen in vielleicht 100‘000 Schritten berechnet werden müssen, statt, wie bei Nvidia, in 1-2 Schritten.
Die Folge ist, dass Applikationen, welche das Framework Equalizer einsetzen auf Systemen mit AMD-Grafikkarten 0 – 3 FPS (Frame per Second) haben. Für eine flüssige Darstellung eines Spieles, Filmes sind jedoch mindestens 25 FPS notwendig.

Weder im Framework I3D noch im Framework Equalizer ist dies als Hardware-Einschränkung vermerkt.
Wir haben deshalb unsere Entwicklungsumgebung mit Nvidia-Grafikkarten ausgestattet.

4.2.1 CAVE – Computer Automated Virtual Environment

Als CAVE wird ein Raum an der Berner Fachhochschule bezeichnet, welcher eine 3D-Umgebung darstellen kann. Er besteht aus folgenden Komponenten:

- 4 Leinwänden
- 8 Beamer (pro Leinwand pro Auge)
- 10 Computer
- 6 Kameras
- 2 Haptic-Geräte

Der CAVE ermöglicht einem Betrachter das Begehen einer 3D-Welt. Mit Hilfe einer 3D-Brille nimmt der Betrachter einen 3D-Effekt war.

4.2.2 Omni Phantom

Das SensAble Phantom Omni gehört zu den haptischen Geräten. Haptische Geräte erlauben dem Benutzer ein aktives Erfühlen einer virtuellen Welt im 3D Raum. Dabei wird versucht die Oberflächenstrukturen, Kräfte, Grössen, Konturen, etc. möglichst realitätsgetreu an den Benutzer weiter zugeben.

Abbildung 6: SensAble Phantom Omni

Im Falle des Phantom Omni‘s reden wir von 6-Freitheitsgraden. Dies bedeutet, die Positionen X/Y/Z, sowie 3 Rotationswinkeln, können genau bestimmt werden.
Kräfte werden dabei auf die 3 Achsen X/Y/Z zurückgegebe. Die Rotationswinkel können keine Kräfte zurückgeben.

4.3 Use-Cases

UseCases sind für ein Spiel schwierig zu finden. Für ein Computerspiel wird im Normalfall ein Storyboard erstellt. Mit diesem wird der gesamte Spielablauf und mögliche Variationen davon beschrieben. Für unser Squash ist ein Storyboard durch die entsprechenden Squash-Regeln gegeben. Anhand von diesen Regeln haben wir bereits im Pflichtenheft unter dem Kapitel "6.12 Use-Cases" verschiedene Use-Cases definiert. Diese werden folgend noch einmal genauer erläutert:

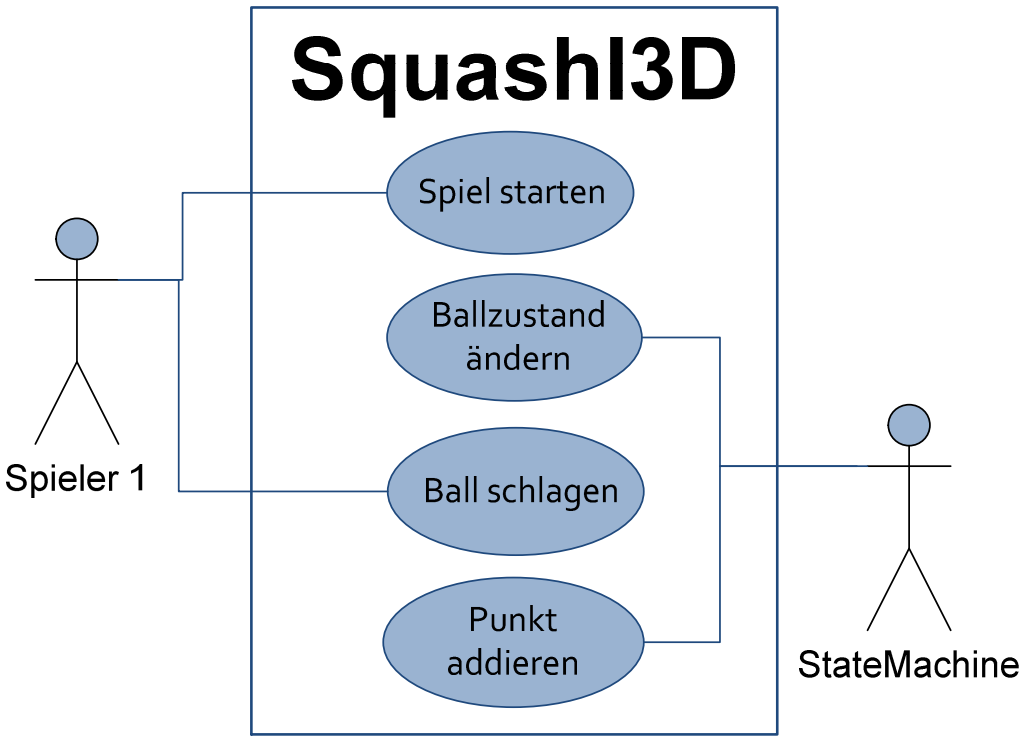


Abbildung 7: Übersicht Test-Cases

4.3.1 Spiel starten

Der Benutzer kann das Spiel mit der Datei "SquashI3D.exe" starten. Um die Applikation im CAVE zu starten, muss die BAT-Datei unter dem Verzeichnis "D:\CaveApps\SquashI3D" ausgeführt werden. Nach dem das Spiel geladen vollständig geladen ist, kann der Benutzer die verschiedenen Einstellungen ändern. Die geänderten Einstellungen werden sofort übernommen. Mit dem Menupunkt "Spiel starten" wird die Squash-Halle geladen und das eigentliche Spiel wird gestartet.

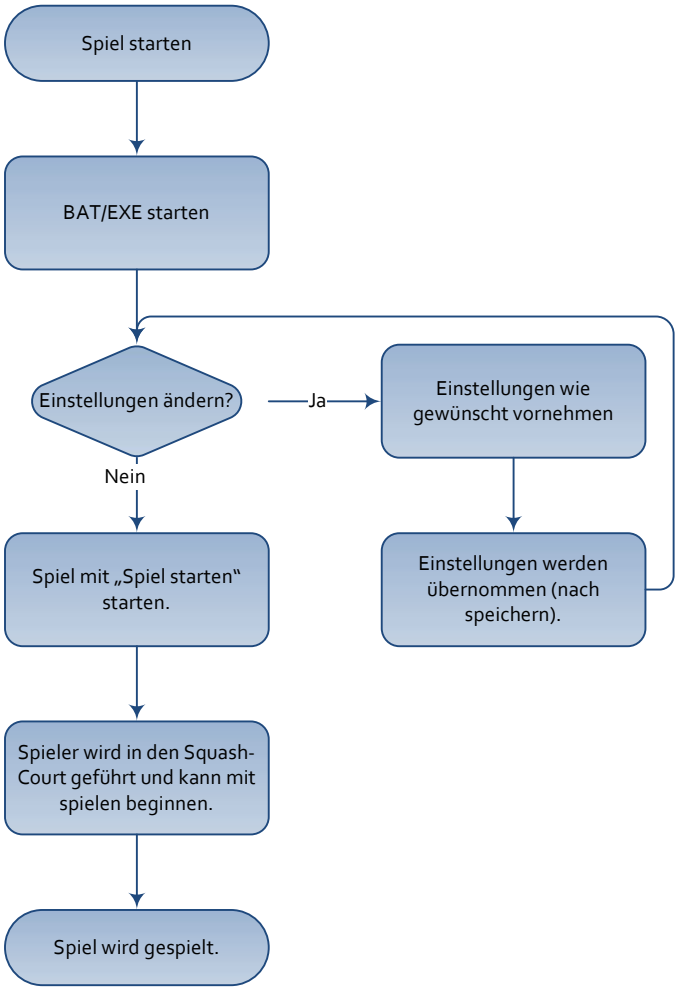
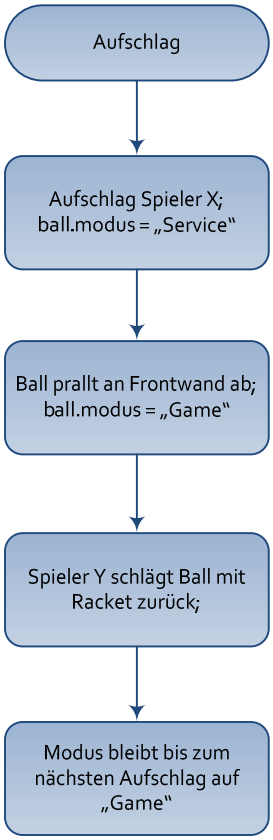


Abbildung 8: Use-Case – Spiel starten inkl. Einstellungen ändern.

4.3.2 Ball-Zustand ändern

Der Aufschlag wird im Squash als einen Spezialfall von den üblichen Ballwechseln gesehen. Beim Aufschlag muss der Ball oberhalb der Aufschlaglinie der Frontwand abprallen. Ansonsten ist es ein ungültiger Aufschlag. Um dies möglichst einfach realisieren zu können, wird der Ball mit einem Modus-Attribut versehen, um beim Aufprall an der Frontwand den entsprechenden Fall überprüfen zu können.

Mit folgendem Use-Case wird beschrieben, zu welchem Zeitpunkt der Ball in welchem Zustand ist. Bei jedem neuen Aufschlag wird der Modus des Balls auf "Aufschlag" gesetzt. Nach dem der Ball die Vorderwand berührt hat, wird der Ball auf den Spiel-Modus gesetzt und bleibt in diesem bis zum nächsten Aufschlag.



Mögliche Ballmodi: „Service“ und „Game“. Somit kann während dem Ballwechsel aufgrund dieses Modus die entsprechende Regel angewendet werden.

```
X   = 0|1
Y   = !X
```

Abbildung 9: Test-Case – Ballmodus

4.3.3 Ball schlagen

Mit den Squash- Regeln ist genau definiert, wann ein Ballwechsel gültig ist und wann dieser durch Missachten einer Regel ungültig wird. Ein Ballwechsel betrifft die Zeitspanne zwischen einem Schlag von Spieler X und einem Schlag von Spieler Y. Dann hat bereits der nächste Ballwechsel angefangen, bis der Ball vom Spieler X wieder zurückgespielt wird.

Eine Regel vom Squash besagt, dass ein Ball vom Racket auf indirekten Weg über die Rück- und Seitenwände an die Vorderwand geschlagen werden muss. Auf diesem darf der Boden nicht berührt werden. Zudem muss der Ball an der Vorderwand oberhalb einer definierten Linie abprallen. Beim Aufschlag ist dies die Aufschlagslinie, welche sich ca in der Mitte der Hallenhöhe befindet, ansonsten ist es die Linie oberhalb des sogenannten Tins, welche nur einige Zentimeter über dem Boden sind.
Auf dem Rückweg darf der Ball den Boden maximal einmal berühren, bevor ihn der Gegenspieler abnehmen und zurückschlagen kann. Die Schlaufe dieser Regel beginnt mit diesem Schlag wieder von vorne.

Wenn der Ball in einer ungültigen Region abprallt, kann mit diesem Use-Case entschieden werden, welcher Spieler den Punkt gewinnt. Tritt ein Fehler während dem Weg des Balles vom Racket des Spieler 1 zur Vorderwand auf, so erhält der Spieler 2 einen Punkt. Ist der Ball korrekt an der Vorderwand abgeprallt und danach passiert ein Fehler, so ist es ein Punktgewinn vom Spieler 1.

Das Diagramm zum Use-Case ist auf der nächsten Seite abgebildet.

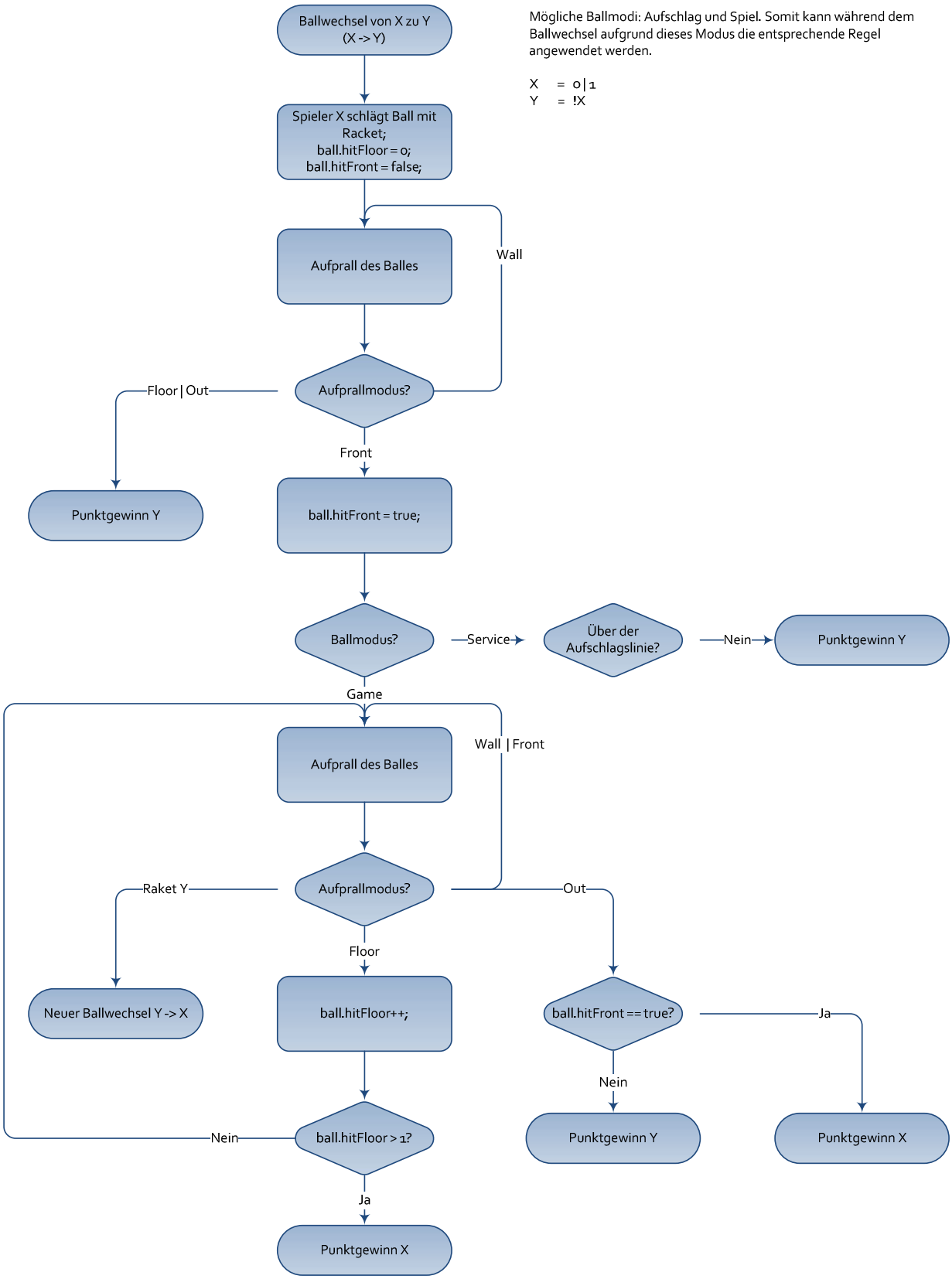


Abbildung 10: Use-Case – Ballwechsel

4.3.4 Punkte addieren

Der Spieler, der in einem Set als erster 21 Punkte hat, gewinnt das Set. Um das Spiel zu gewinnen, muss der Spieler insgesamt 3 Sets gewinnen.

In diesem Use-Case bilden wir die Verwaltung der Punkte für ein einzelnen Set ab. Aus dem Use-Case "<LINK ZU USECASE Ball Schlagen>" wird gelesen, welcher Spieler der Punkt erhält. In dem folgenden Diagram wurde einfachheitshalber der entsprechende Spieler als X repräsentiert. Im Squash ist es zusätzlich so, dass der Spieler den Aufschlag erhält, der den letzten Punkt gewonnen hat.

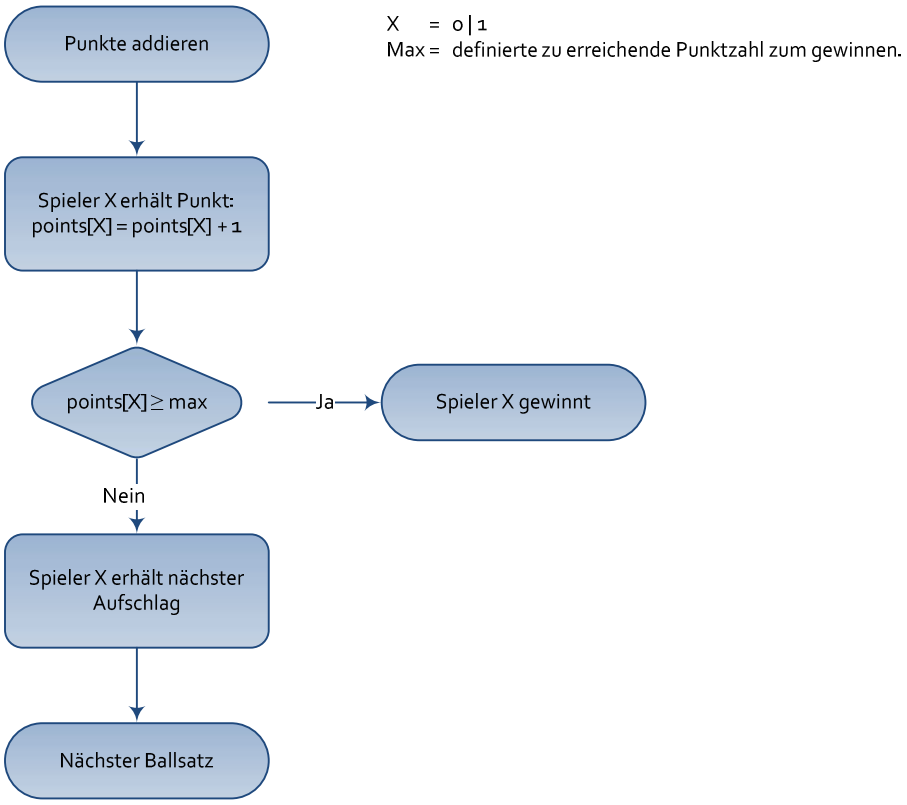


Abbildung 11: Test-Case – Punktegewinn

4.4 Spielregeln

Die Spielregeln sind bereits im Pflichtenheft unter dem Kapitel 4.1 beschrieben worden. Hier folgt der Vergleich, welche Regeln wir wie umgesetzt haben.

4.4.1 Die Squash Spielregeln

Im reelen Squash gelten folgende Regeln:

Aufschlag:

Der Aufschlag wird immer aus einem der beiden Aufschlagfelder ausgeführt. Dabei muss der Aufschläger mit mindestens einem Fuß im Aufschlagfeld stehen. Nach dem Aufschlag muss der Ball die Stirnwand oberhalb der Aufschlaglinie treffen und auf der anderen Seite, im Viertel des Gegners aufkommen. Beim ersten Aufschlag kann der Spieler die Aufschlagsseite frei wählen, muss dann aber die Seite nach jedem Punktgewinn wechseln. Wenn der Aufschlagende den Ballwechsel verliert, erhält der Gegenspieler das Aufschlagrecht. Im Gegensatz zu anderen Racketsportarten darf der Ball beim Squash für den Aufschlag auch mit dem Schläger angeworfen werden.

Ballwechsel

Der Ball muss nach jedem Schlag auf direktem oder indirektem Weg die Vorderwand berühren. Als indirekt gilt ein Weg über Seiten- und Rückwand. Danach darf der Ball nicht mehr als einmal auf dem Boden, jedoch beliebig oft auf die Rückwand oder Seitenwände auftreffen, bevor er vom Spielpartner zurückgeschlagen wird. Ein Ball gilt im „Aus“, wenn er die Wände oberhalb der dort angebrachten roten Begrenzungslinien, die Begrenzungslinie selbst oder das Tin berührt.

Zählweise

Es gilt, dass ein Spiel über drei Gewinnsätze geht, d. h. der Spieler, der als erster drei Sätze gewinnen kann, entscheidet das Spiel für sich. Es wird jeder Punkt gezählt, egal wer das Aufschlagrecht hatte. Für den normalen Satzgewinn benötigt der Spieler 11 Punkte. Beim Stand von 10:10 wird ein Tie-Break gespielt. Hier gewinnt derjenige Spieler, welcher zuerst 2 Punkte Vorsprung hat (z. B. 13:11, 19:17 usw.).

4.4.2 Regeln im SquashI3D

Im Pflichtenheft wurde definiert, dass für unser Spiel folgende Regeln gelten:

- Der Ball muss nach jedem Schlag auf direktem oder indirektem Weg die Vorderwand berühren. Als indirekt gilt ein Weg über Seiten- und Rückwand.
- Der Ball darf auf dem Weg von der Vorderwand zum Spieler nicht mehr als einmal den Boden berühren.
- Ein Ball gilt als im „Aus“, wenn er die Wände oberhalb dort angebrachter roter Begrenzungslinien, die Begrenzungslinie selbst oder das Tin berührt.
- Die Rückwand gilt als „Aus“.

Eine weitere Anpassung kommt aus der Optimierungsphase. Um den Spielspass zu erhöhen, werden die Sätze auf 21 gespielt. Die Spieler können unendlich viele Sätze spielen.

4.5 Software Architektur

Die Architektur kann vereinfacht, wie folgt dargestellt werden.

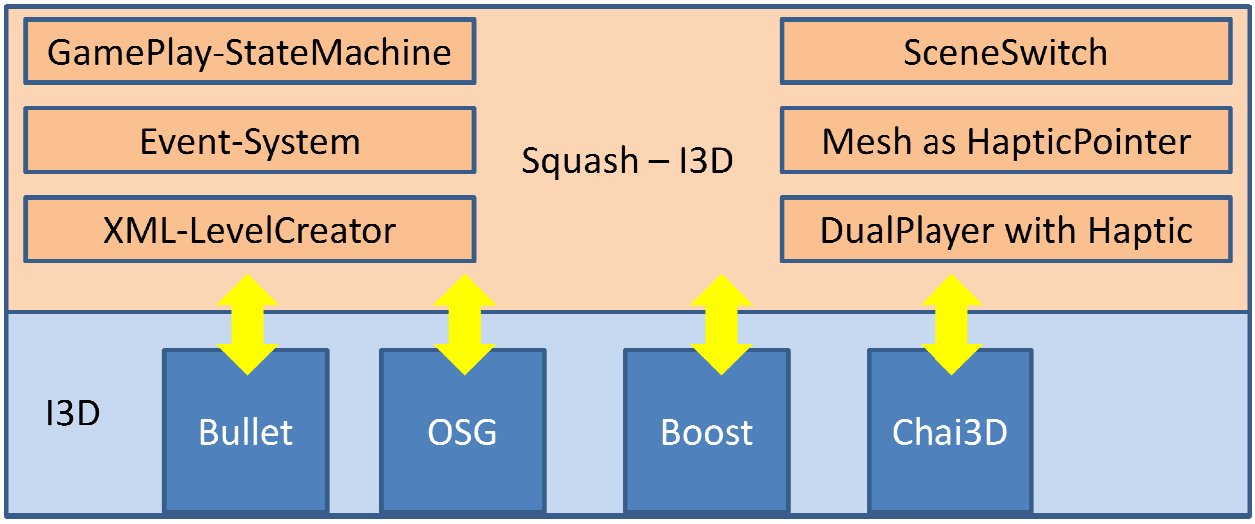


Abbildung 12: Software Architektur

SquashI3D verwendet als Framework I3D. I3D seinerseits baut auf folgenden Frameworks auf.

Framework	Beschreibung
I3D	Framework, welches an der BFH entwickelt und eingesetzt wird. Mit Hilfe des Frameworks wurde die CAVE-Integration umgesetzt.
Bullet	ist eine Physik-Engine, welche im I3D-Framework integriert wurde.
OSG	OpenSceneGraph erlaubt das Aufbauen einer 3D-Welt mit Hilfe eines Graphen. Dieses Framework wurde im I3D integriert.
Boost	umfasst viele Unterbibliotheken, welche Plattform-unabhängig

	verwendet werden können.
Chai3D	ist ein Haptic-Framework, welches ins I3D-Framework integriert wurde.

Tabelle 5: Framework-Komponenten

Folgend sind die Key-Feature von SquashI3D beschrieben.

Feature	**Beschreibung**
Gameplay-BallStateMachine	Die Squash Regeln für SquashI3D wurden mit Hilfe des State-Patterns abgebildet. Der aktuelle Spielzustand wird damit als Status ausgedrückt. Dies hat grosse Vorteile in der Synchronisation der Nodes.
Event-System	Die Synchronisation der verschiedenen Nodes geschieht im I3D-Framework über das FrameData-Objekt. Dieses Objekt hat für jeden spezifischen Wert, welcher synchronisiert werden muss, ein spezifisches Attribut. Für SquashI3D ist dies nicht praktisch. Deshalb entwickelten wir ein Event-System, das Events über das FrameData-Objekt versendet. Diese Events stellen die Synchronisation der Nodes sicher.
XML-LevelCreator	Der Aufbau von Szenen kann ebenfalls dynamisch gehalten werden. Mit dem XML-LevelCreator haben wir ein XML-Format geschaffen, mit dessen Hilfe sehr schnell neue Welten erstellt werden können.
SceneSwitch	Da wir uns zum Ziel gesetzt haben, mit mehreren SquashI3D Räumen arbeiten zu wollen, wollten wir einen On-the-Run Switch ermöglichen. Dies haben wir mit der Kapselung des SceneManagers erreicht. Gekapselt wird das SceneManager durch das SceneSetup.
Mesh as HapticPointer	Bisher wurden im CAVE die haptischen Geräte als Kugel dargestellt. Für SquashI3D haben wir ein Racket designed, welches eher zum Thema Squash passt, als eine Kugel.
DualPlayer with Haptic	Im CAVE waren haptische Geräte bisher nur als Einzelgerät einsetzbar. Wir haben dies um ein weiteres Geräte erhöht. 2-Geräte ist die maximale Anzahl, die das Haptik-Framework Chai3D unterstützt.

Tabelle 6: Framework-Komponenten

4.6 Package Diagramm

Das Package-Diagramm soll die Gliederung unserer verschiedenen Packages im Projekt aufzeigen. Zusätzlich können so die verschiedenen Abhängigkeiten der Packages untereinander besser visualisiert werden.

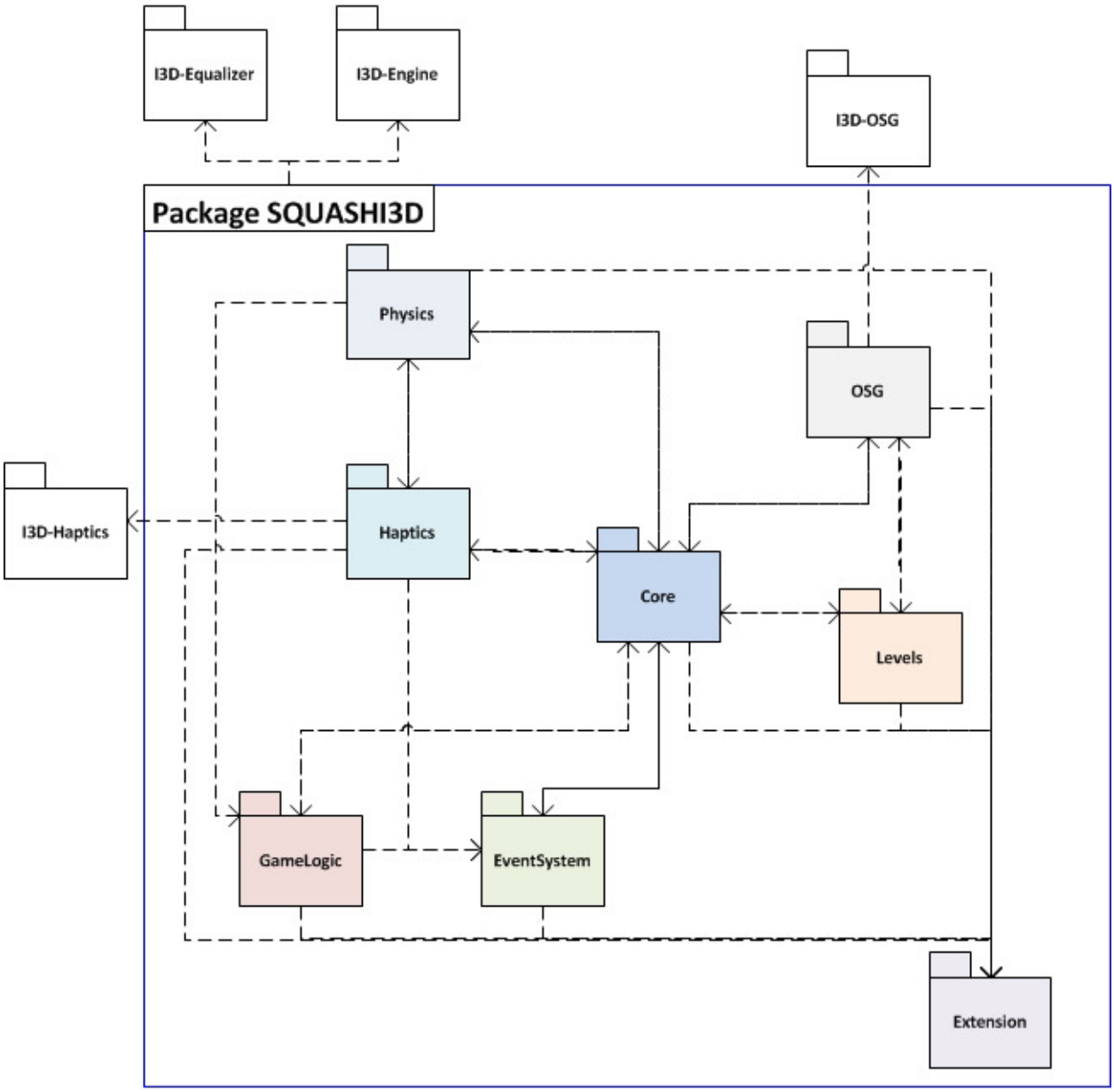


Abbildung 13: Package-Diagramm

Package SquashI3D	Beschreibung
Core	Bezeichnet das Package, welches alle Daten synchronisiert. Es dient als Drehscheibe für alle Informationen.
EventSystem	Dient dem Datenaustausch zwischen den verschiedenen Computern.
Extension	Enthält verschiedene Hilfsklassen
GameLogic	Verwaltet die gesamt Logik für SquashI3D.
Haptics	Steuert den Zugriff auf die Haptic-Komponenten. Das Package Haptic ist eng mit dem Physics-Package verbunden, um die Gerät Position möglichst korrekt halten zu können.
Levels	Erstellt den graphischen Aufbau unseres Spiels. Es fügt die verschiedenen graphischen Element zusammen.
OSG	Enthält für den OpenSceneGraph spezifische Funktionen.
Physics	Dient der Berechnung der Physik.

Tabelle 7: Package Beschreibungen

4.7 System Design

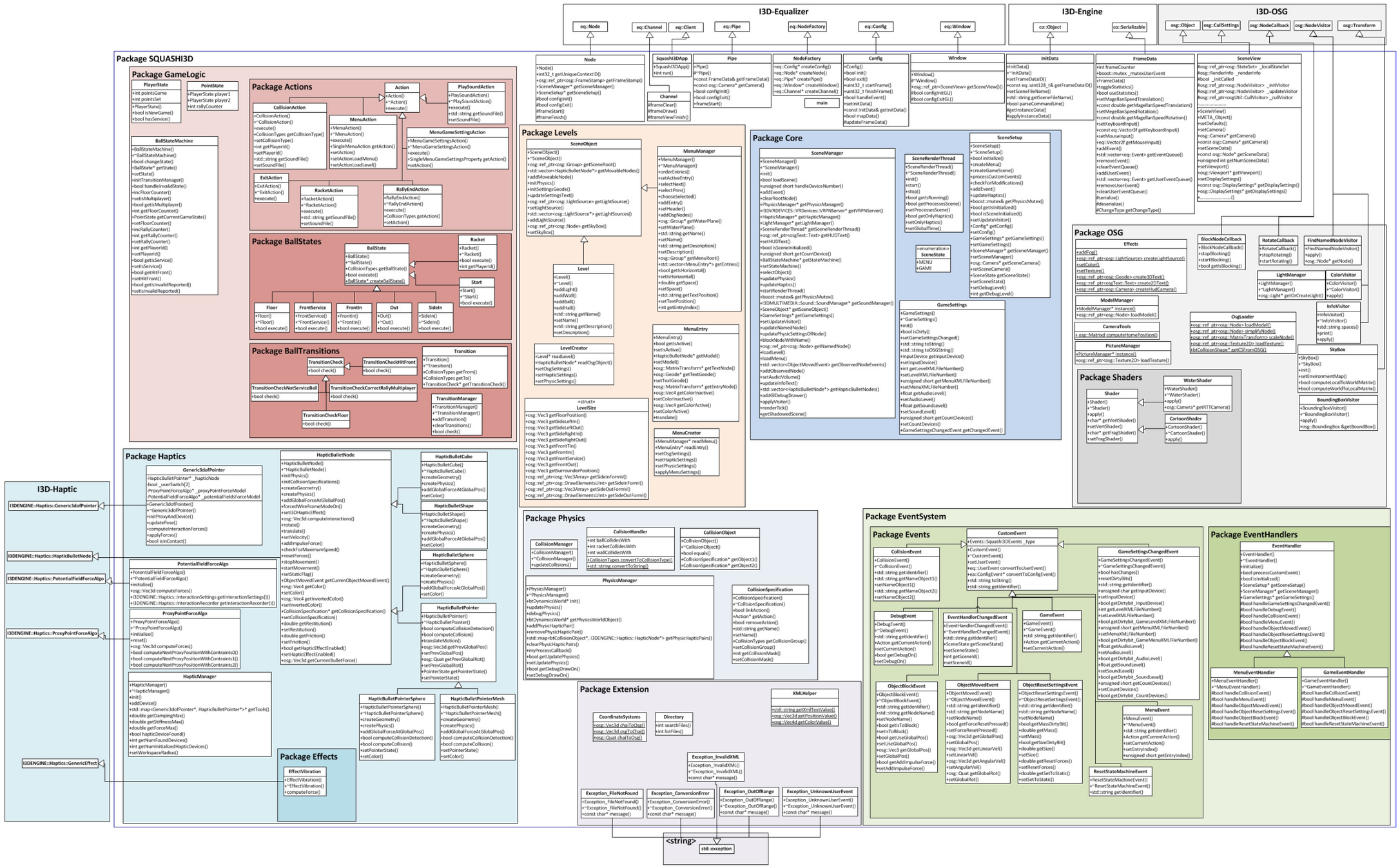


Abbildung 14: Systemarchitektur

Die Systemarchitektur ist komplex und als Ganzes schwer zu verstehen. Unten aufgeführt sind die wichtigsten Bereiche speziell erklärt.

4.7.1 Klassen

Package: SQUASHI3D	
Name	**Verwendungszweck**
SquashI3DApp	Enthält hauptsächlich den Run loop.
Config	Die Config-Klasse ist eine spezielle Klasse, welche sogenannte ConfigEvent‘s verarbeitet. Diese ConfigEvent-Objekte sind durch das Framework Equalizer vorgegeben. Die Config-Klasse verarbeitet die Events (ConfigEvent als CustomEvents) und überträgt die Informationen an das FrameData. Das FrameData wird anschliessend an die anderen Nodes übermittelt.
FrameData	Ist das Objekt, welches an alle Nodes (Computer) versendet wird. Es enthält alle Informationen, welche synchronisiert werden müssen.
Node	Die Node-Klasse leitet von der Equalizer Node ab. Sie enthält zusätzliche Referenzen auf das SceneSetup. Pro Computer wird eine Node initialisiert.
Pipe	Die Pipe nimmt das FrameData-Objekt entgegen. Die darauf enthaltenen Informationen werden entweder direkt verarbeitet oder an das SceneSetup zur Verarbeitung weitergegeben.
Package: Core	
GameSettings	Enthält Daten über die aktuellen Spieleinstellungen: - Einstellungen XML-Dateiname - Aktueller InputDevice Handler - XML-Level-Konfiguration XML-Dateiname - XML-Menu-Konfiguration XML-Dateiname - Audio Lautstärke - Sound Lautstärke
SceneManager	Der SceneManager enthält die aktuelle Szene. Er verwaltet den OpenSceneGraph (Aufbau der aktuellen Szene), steuert den PhysicManager und den HapticManager.
SceneSetup	Die SceneSetup-Klasse enthält die Funktionalität um eine Szene komplett auszutauschen. Es verwaltet sowohl den aktuellen EventHandler, die GameSettings, sowie den SceneManager. Das SceneSetup führt das Laden einer neuen Szene durch und setzt dieses auf dem SceneManager.

SceneRenderThread	Ist für die ständige Aktualisierung und Rendern der geladenen Haptic-Geräte zuständig. Zusätzlich wird durch diesen die Physik aktualisiert.
Package: EventSystem	
Package: Events	
CustomEvent	Kapselt die ConfigEvent-Klasse von Equalizer. Vereinheitlicht den Weg, wie mit ConfigEvent in SquashI3D gearbeitet wird. Es fungiert sowohl als Builder von ConfigEvents, wie auch als Interpreter.
Package: EventHanlders	
EventHandler	Ist die Basis Klasse für die Event-Handler‘s von SquashI3D. Sie erstellt aus den ConfigEvent wieder CustomEvents und verarbeitet diese.
Package: GameLogic	
BallStateMachine	Die Squash Regeln wird mit Hilfe des State-Patterns abgebildet. Die BallStateMachine dient dazu, diese Regeln in unserem System abzubilden und somit die Gültigkeit des aktuellen Ball-Status zu bewerten.
Package: BallTransitions	
Transition	Stellt einen erlaubten Statuswechsel von einem gegebenen Status zu einem Neuen dar. So kann die BallStateMachine mit Regeln initialisiert werden, in welcher nur bestimmte Übergänge erlaubt sind.
TransitionCheck	Ist eine zusätzliche Überprüfung für eine Transition. Diese wird nebst dem erlaubten Wechsel geprüft, ob der Übergangswechsel erlaubt ist oder nicht. So kann die BallStateMachine flexibel um Verzweige der Übergänge erweitert werden.
TransitionManager	Verwaltet alle gültigen und erlaubten Übergänge. Die BallStateMachine kann den TransitionManager bei einem Übergangswechsel abfragen, ob von aktuellen in den neuen Status gewechselt werden darf oder nicht.

Package: Haptics	
Generic3dofPointer	Definiert, wie mit einem haptischen Gerät interagiert wird. Auf dieser Klasse sind alle Werte definiert, die den Zugriff auf ein haptisches Gerät ermöglichen. Mit Hilfe dieser Klasse kann Bullet (Physik) die Werte aktualisieren, welche der Benutzer als haptisches Feedback wahrnimmt. Wurde erstellt, um die Arbeit mit komplexeren Objekten als Kugeln zu ermöglichen.
HapticManager	Verwaltet die ganze Haptik. Mit dem Initialisieren prüft der Haptik-Manager, ob der Computer über haptische Geräte verfügt. Falls ja, können diese initialisiert werden. Falls nein, wird der HapticManager deaktiviert.
HapticBulletNode	Ist die Verbindung zwischen Bullet, OSG und Chai3D im SceneGraph.
HapticBulletPointer	Ist eine abstrakte Klasse welche als Haptic-Pointer im 3D-Raum funktioniert. Dieser beinhaltet alle relevanten Funktionalitäten und Schnittstellen, um mit einem Haptic-Gerät zu interagieren.
Package: Levels	
SceneObject	Das SceneObject stellt eine komplette Scene im OSG für unser Spiel dar. Alle wichtigen Listen und OSG-Knoten sind in dieser Klasse implementiert, um im SceneManager geladen und dargestellt zu werden.
MenuManager / Level	Der MenuManager und das Level leiten beide vom SceneObject ab und können im SceneManager geladen werden. Der MenuManager hat zusätzliche Methoden um im Menu zu navigieren, deshalb wurde die Klasse MenuManager benannt.
LevelCreator / MenuCreator	Der LevelCreator sowie der MenuCreator lesen entsprechende XML-Dateien ein um daraus ein Level/Menu zu generieren, das alle nötigen OSG, Bullet und Haptic-Eigenschaften beinhaltet.
Package: Physics	
PhysicsManager	Der PhysicsManager verwaltet die gesamte physikalische Welt sowie derer Vorgänge. Dazu wird das in I3D integrierte Bullet verwendet. Ebenfalls werden pro Frame die Kollisionen erkannt und an den CollisionManager gesendet.
CollisionManager	Mit dem CollisionManager werden die verschiedenen Kollisionen getriggert. Die Kollisionen werden pro Frame durch den PhysicsManager aktualisiert. Der CollisionManager erkennt dann alle neuen Kollisionen und kann entsprechende Aktionen dazu ausführen.

Package: OSG	
OSGLoader	Lädt verschiedene Dateien vom Filesystem in den SceneGraph von OSG.
PictureManager	Verwaltet geladene Bilder. Alle zu ladenden Bilder werden gespeichert, damit diese bei einem weiteren oder erneuten Gebrauch nicht wieder geladen werden müssen. Dies erspart langes Warten bei erneutem Level-Laden.
ModelManager	Verwaltet geladene Modelle. Alle zu ladenden Modelle werden gespeichert, damit diese bei einem weiteren oder erneuten Gebrauch nicht wieder geladen werden müssen. Dies erspart langes Warten bei erneutem Level-Laden.
Effects	Kann verschiedene Effekte im SceneGraph erstellen, wie zum Beispiel Nebel oder Farbwechsel eines OSG-Knoten.

Tabelle 8: Wichtige Klassen SquashI3D

4.7.2 Equalizer

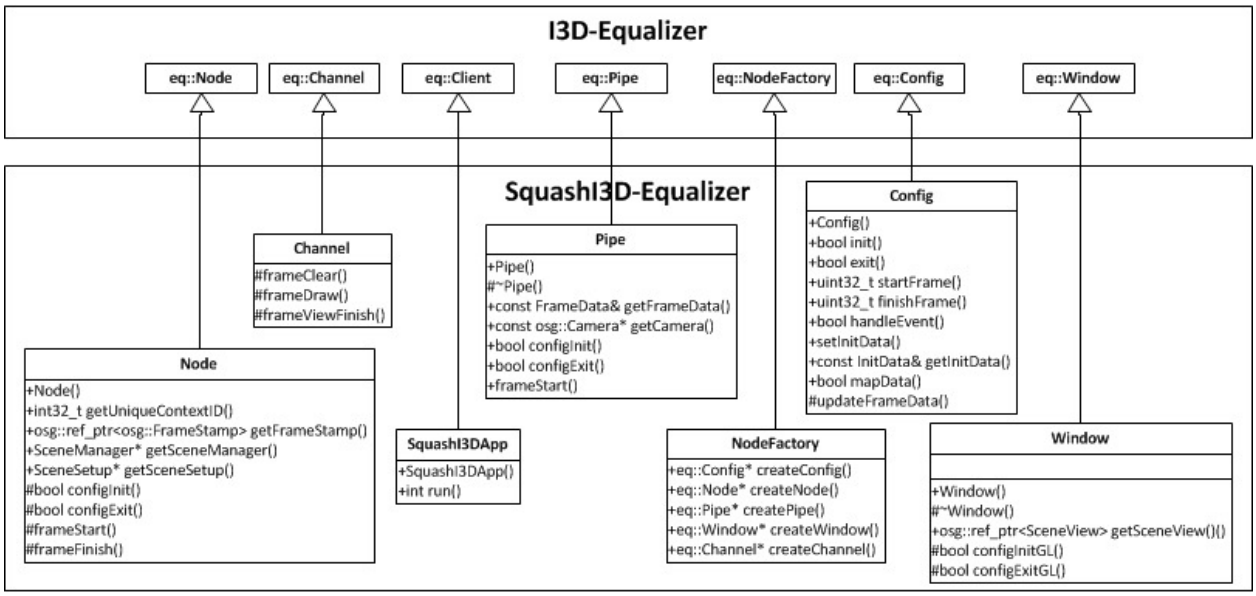


Abbildung 15: Equalizer

Die obengenannten SquashI3D-Equalizer Klassen werden vom Equalizer abgeleitet, um gewisse Funktionalitäten überschreiben/ beeinflussen zu können. Die Node spielt dabei eine zentrale Rolle, da dort die Referenz auf das SceneSetup gesetzt wird.
Node gibt es pro Computer genau einmal. Die Pipe hingegen, existiert pro Grafikausgang. Dies bedeutet, auf einem Computer mit zwei Bildschirmen gibt es ein Node-Objekt und zwei Pipe-Objekte.
Diese Erkenntnis ist von Bedeutung, da die Pipe das FrameData-Object (siehe 4.7.1 Klassen FrameData) auspackt und verarbeitet. Das bedeutet, pro Pipe wird ein FrameData versendet. Dies gibt Probleme bei Informationen, welche pro Computer nur einmal relevant sind. Siehe 5.1 FrameData.
Equalizer als Framework ist nicht sehr gut dokumentiert. Viele Funktionen sind nicht ausreichend beschrieben. Bei vielen Schwierigkeiten haben wir deshalb nach „Try&Error“ die Funktionen ausgetestet. Dies nahm viel Zeit in Anspruch.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.3.1 Framework I3D

4.7.3 SceneSetup

Das SceneSetup dient als Fassade für die Equalizer-Klassen. Dies wurde so implementiert, um die Referenzen in den Equalizer-Klassen gering zu halten. Mit Hilfe der SceneSetup-Klasse realisierten wir das Feature „Menuführung“. Damit können wir Szenen (Menu oder Spielräume) zur Laufzeit austauschen.

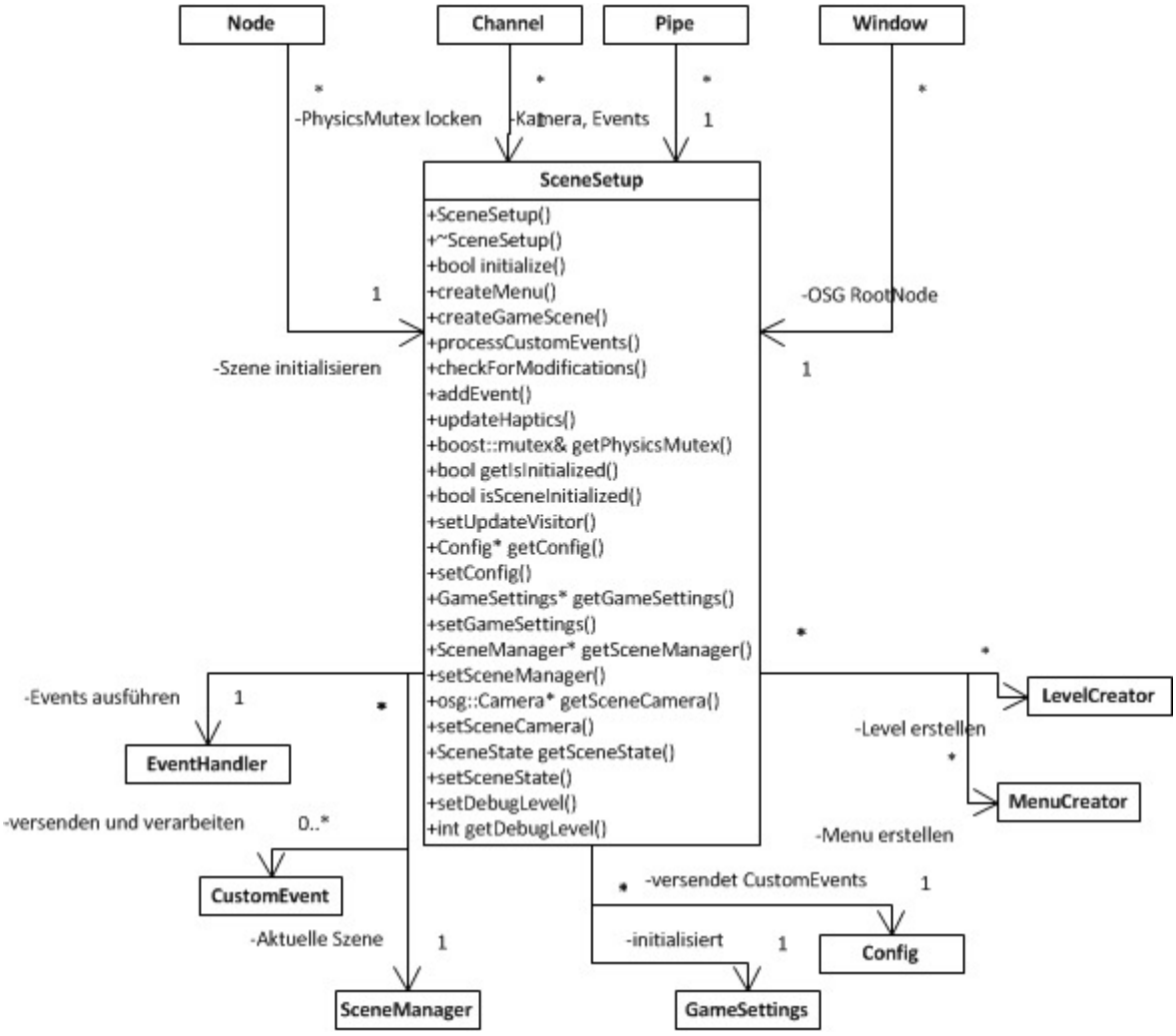


Abbildung 16: SceneSetup

Das SceneSetup erstellt die Rahmenbedingungen für die ganze SquashI3D-Applikation. Alle Synchronisierungen (siehe 4.7.5 Event-System) werden über diese Klasse gemacht.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.5.14 Menuführung
- 6.6.3 VSP2: Verschiedene Räume

4.7.4 SceneManager

Die SceneManager-Klasse ist die zweite grosse Klasse von SquashI3D. Alles was mit dem aktuellen Applikationszustand zu tun hat, ist auf dem SceneManager untergebracht. Er verwaltet folgende Bereiche:

- Haptic
- OSG-Tree
- Szenenbeleuchtung
- SoundManager
- Physik
- StateMachine von SquashI3D

Der SceneManager entscheidet beim Initialisiern, ob die Haptic zur Verfügung steht oder nicht. Das bedeutet, ist ein haptisches Gerät vor Starten des Spieles nicht angeschlossen, kann nachträglich kein neues Gerät mehr hinzugefügt werden. Die Applikation muss neu gestartet werden.
Findet der SceneManager ein oder mehrere haptische Geräte, so initialisert er die Physik und die Haptic. Die Physik wird ausschliesslich zusammen mit der Haptic initialisiert.
Im CAVE bedeutet dies, dass nur ein Computer die Physik berechnet. Dies reduziert die Flut von Kollision-Events (siehe 4.7.5.1 CustomEvents), welche sonst entstehen würde.

Auf der nächsten Seite wird die Klasse im Detail gezeigt.

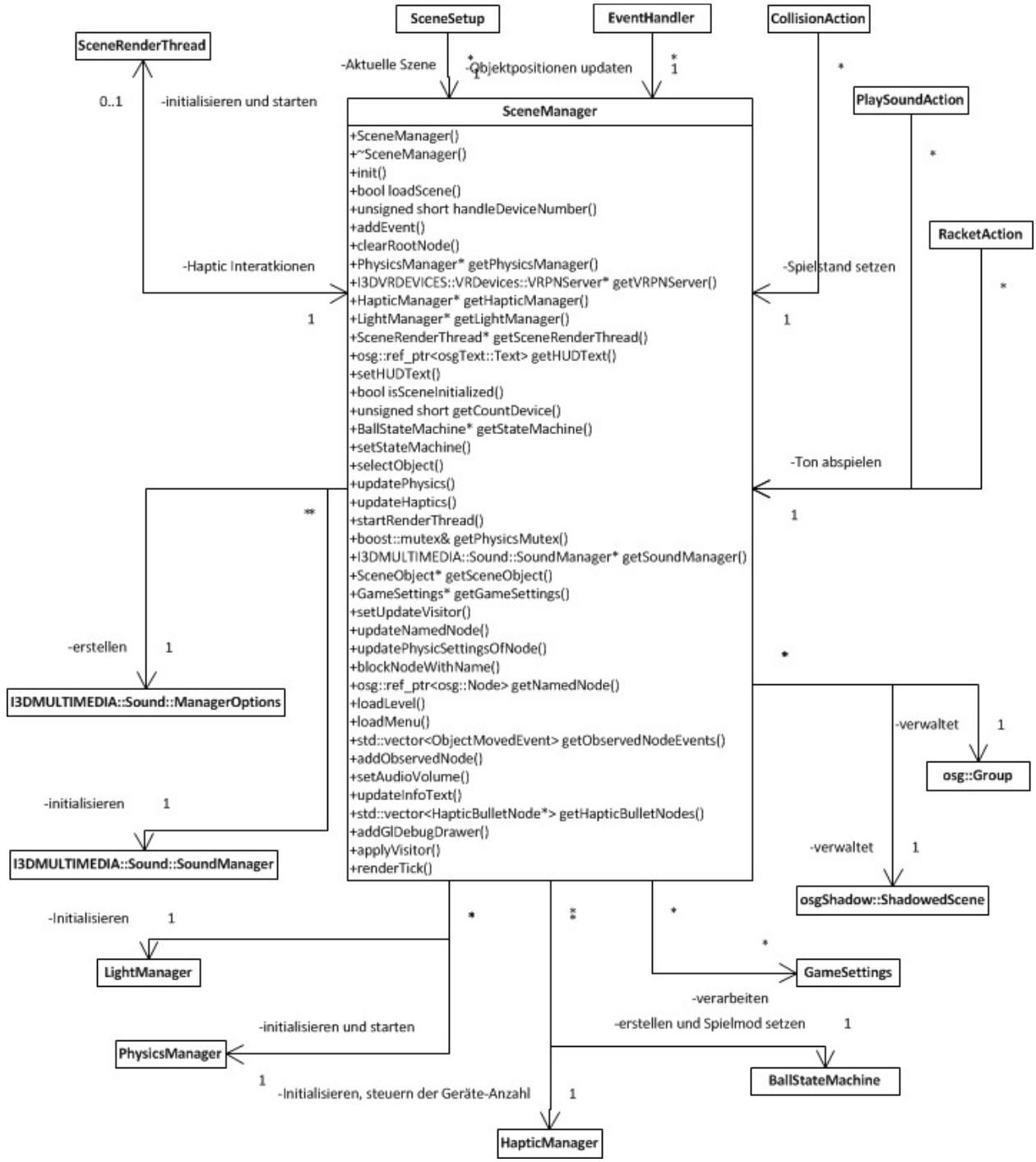


Abbildung 17: SceneManager

Alle Änderungen an der Szene laufen über den SceneManager.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.3.2 Level-Design mit OpenSceneGraph
- 6.4.3 Rendering Basic
- 6.4.4 Rendering Advanced

4.7.5 Event-System

UserEvents werden wie folgt gekapselt:

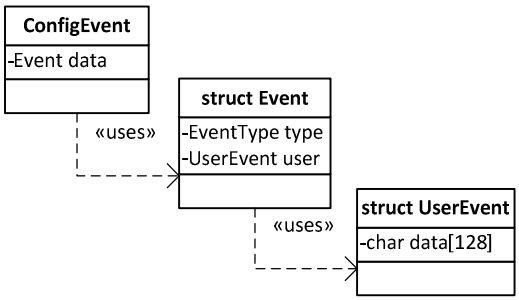


Abbildung 18: Kapselung UserEvent

Dieser Aufbau der ConfigEvents ist vorgegeben durch das Equalizer-Framework. Dabei gilt es folgende Vorgaben einzuhalten.

- Custom-Events müssen vom EventType: USER sein.
- Maximal können 128 Byte übertragen werden.

Grundsätzlich übermittelt I3D-Framework keine ConfigEvent über das FrameData. In SquashI3D führt das FrameData eine Liste von ConfigEvents mit und übermittelt diese. Die ConfigEvents werden auf dem Config-Objekt gesammelt und pro FrameData übermittelt. Auf der Pipe werden die FrameData-Informationen verarbeitet. Ebenfalls die ConfigEvents des SquashI3D.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.3.1 Framework I3D
- 6.5.11 Input – Interaktion Hatpic
- 6.5.5 Spielprinzip – Verschiedene Räume

Das Event-System bildet die Basis von SquashI3D. Ohne dieses können keine Aktionen ausgeführt werden.

4.7.5.1 CustomEvents

Die Klasse CustomEvents kapselt den ConfigEvent. Das Zusammenstellen der ConfigEvent‘s wird in dieser Klasse erledigt. Ableitende Events müssen lediglich die serialize/ deserialize und attributesToString-Funktionen überschreiben.
Die Funktion convertToConfigEvent gibt einen gültigen ConfigEvent zurück, der so an die Config und von da, über das FrameData, versendet werden kann.
Für das Serialisieren und Deserialisieren standen verschiedene Möglichkeiten zur Verfügung. Das Problem aller Lösungen war, dass die Begrenzung von 128 Byte nicht überprüft werden konnte. Deshalb haben wir uns entschlossen das Serialisieren und Deserialisieren von „Hand“ zu machen.

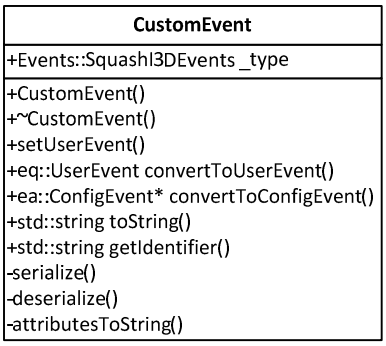


Abbildung 19: CustomEvents-Klassenbeschreibung

Die CustomEvent-Klassen fungieren als Builder (Builder-Pattern) und Interpreter für ConfigEvent‘s/ UserEvent‘s.
In SquashI3D kennen wir folgende Events:

- CollisionEvent
- DebugEvent
- EventHandlerChangedEvent
- GameSettingsChangedEvent
- MenuEvent
- ObjectBlockEvent
- ObjectMovedEvent
- ObjectResetSettingsEvent

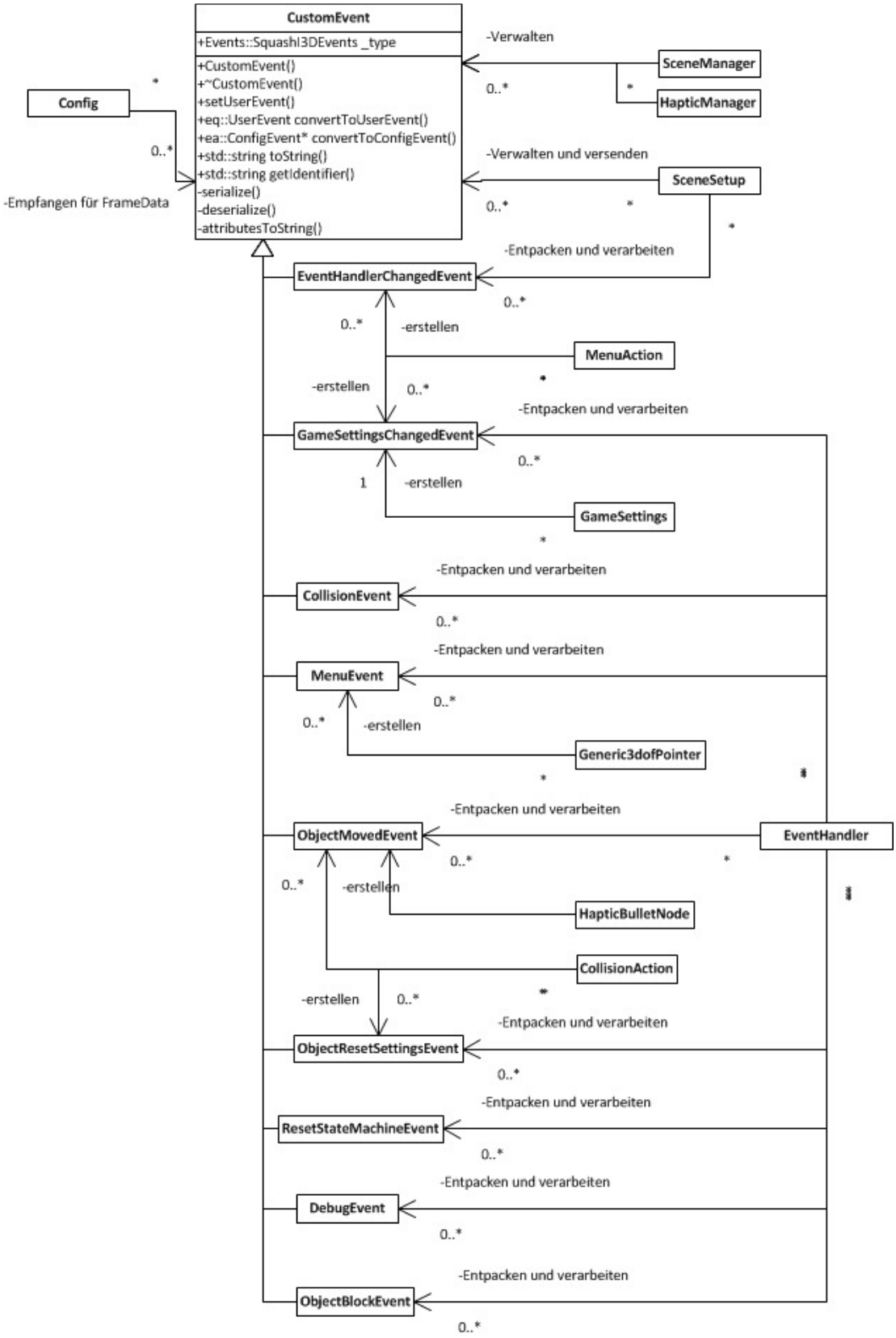


Abbildung 20: CustomEvents

Event-Klassen	Beschreibung
CollisionEvent	Bullet generiert sogenannte Kollisionen. Dies ist der Fall, wenn der Ball z.B. auf dem Boden aufschlägt. Diese Informationen sind relevant für die BallStateMachine (siehe 4.7.7 GamePlay - BallStateMachine). Da dies ebenfalls über den Equalizer synchronisiert werden muss, müssen diese als Event an alle Nodes versendet werden.
DebugEvent	
EventHandlerChangedEvent	Dies ist ein spezieller Event, welcher eine EventHandler-Wechsel provoziert. Siehe 0 EventHandler-Wechsel
GameSettingsChangedEvent	Enthält alle Änderungen der GameSettings. Wird auf den GameSettings auf einer Node etwas geändert, werden die anderen Nodes über diesen Event synchronisiert. Es werden jeweils nur die Änderungen transferiert.
MenuEvent	Ist ein Event, welcher auf Grund einer Taste oder Knopf (Haptik-Geräte) erstellt und verarbeitet wird.
ObjectBlockEvent	Blockiert ein bestimmtes Objekt an einer bestimmten Position. Mit dem gleichen Event kann die Blockade wieder aufgehoben werden.
ObjectMovedEvent	Wird verwendet um die Physik- und Haptik-Updates zu synchronisieren. Dies kommt für den Ball und die Rackets in Frage. Dabei wird immer die genaue Position übergeben und nicht das Delta.
ObjectResetSettingsEvent	Erlaubt das Setzen verschiedener Bullet-Eigenschaften auf einem Objekt. Dies wird gebraucht, falls ein Objekt nicht mehr den physikalischen Gesetzen gehorchen soll.

Tabelle 9: CustomEvents – Beschreibung

Verwalten der Events

Abbildung 21: Verwalten der Events

Die Verwaltung der Events geschieht in einer Liste. Unter Verwaltung von Events wird das Sammeln von, in diesem Zyklus erstellten, Events verstanden. Ein Zyklus wird durch den Takt des Sendens von FrameData-Objekten definiert.
Das SceneSetup verwaltet die Liste von CustomEvent-Objekten, welche versendet werden müssen (siehe 4.7.5.3 Sequenzdiagramm Event-Cycle).

Der SceneManager, sowie der HapticManager enthalten einen Zeiger auf diese Liste. Damit können auf diesen Objekten ebenfalls Events direkt zur Liste hinzugefügt werden.
Da die CustomEvents aus unterschiedlichen Threads erstellt werden, ist die Liste durch einen Mutex geschützt.

4.7.5.2 Event-Handlers

Der Event-Handler verwendet ein Factory-Pattern, um aus den ConfigEvents wieder SquashI3D-Events zu machen. Anhand des ersten Char im Array entscheidet der EventHandler, was für ein Typ von CustomEvent es ist. Anschliessend wird die ConfigEvent entpackt und die entsprechende CustomEvent-Klasse verwandelt.

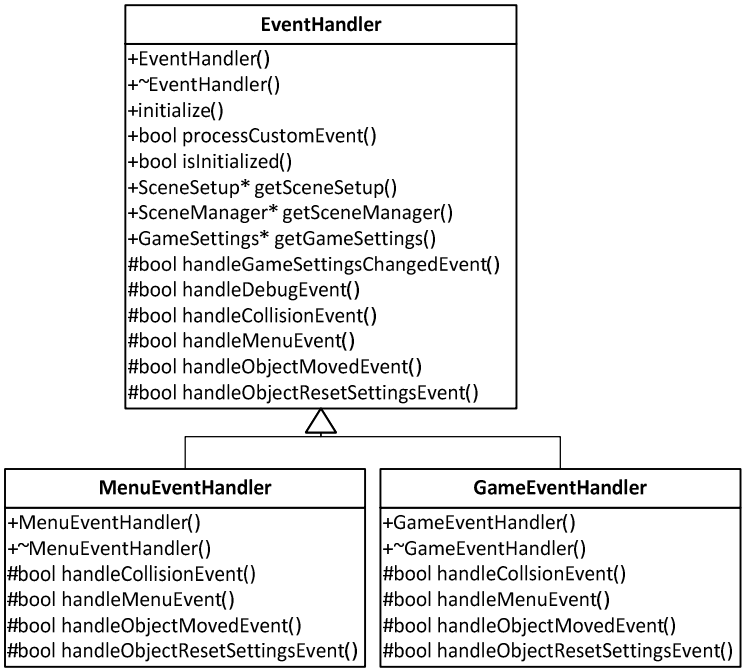


Abbildung 22: Event-Handlers

MenuEventHandler und GameEventHandler unterscheiden sich in der Verarbeitung der folgenden Events:

	MenuEventHandler	**GameEventHandler**
handleCollsionEvent()	Entscheidet, welcher Menu-Punkt ausgewählt ist.	Verwaltet die BallStateMachine.
handleActionEvent()	Steuert die Menu-Aktionen	Ruft das Menu auf oder setzt den Ball zurück.
handleObjectMovedEvent()	Steuert die Position der Rackets	Steuert die Position des Balles und der Rackets.

Tabelle 10: Event-Handlers Unterschied

4.7.5.3 Sequenzdiagramm Event-Cycle

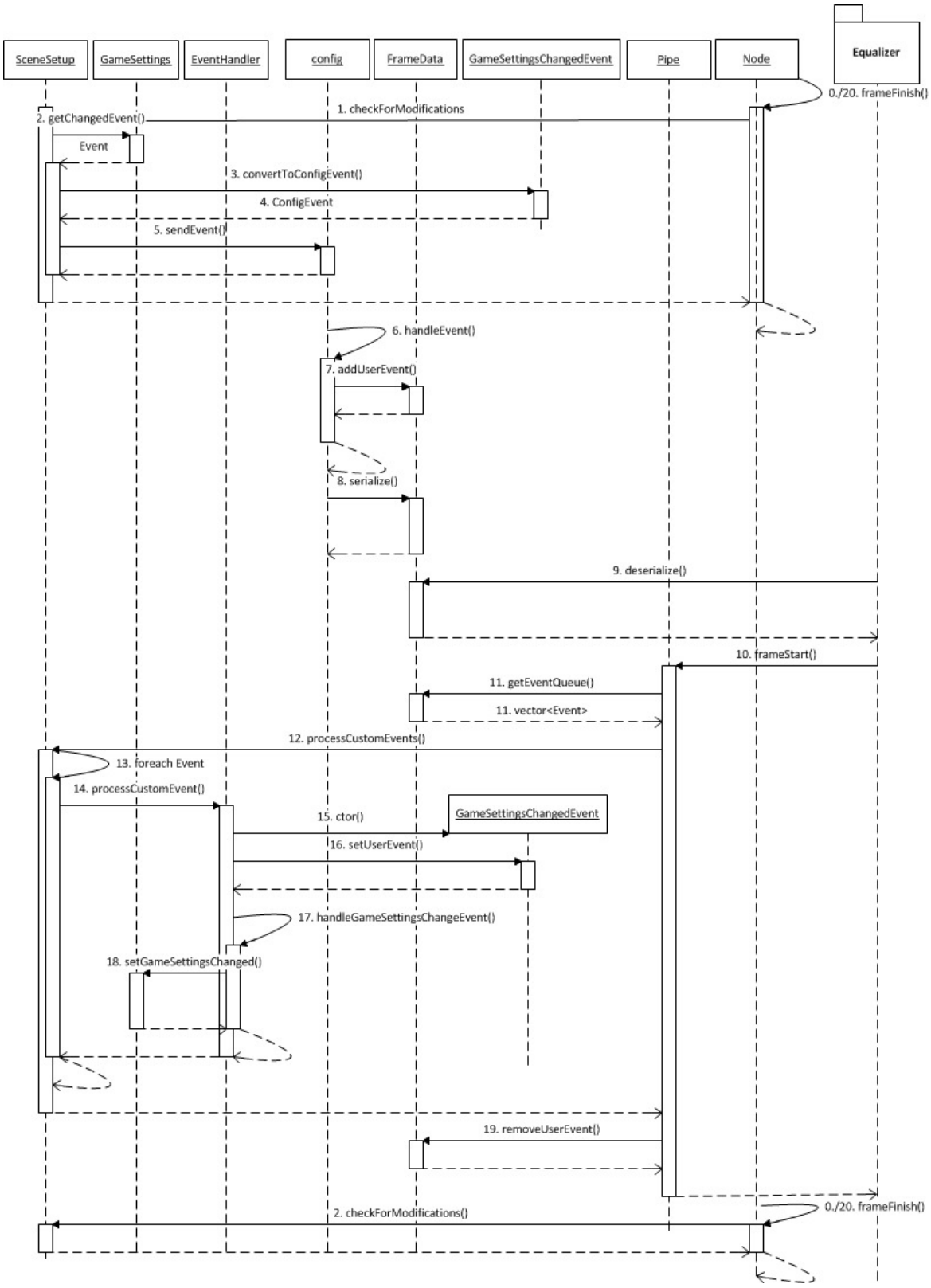


Abbildung 23: Sequenzdiagramm GameSettingsChangedEvent

Das Sequenzdiagramm zeigt nicht den ganzen Ablauf, da dafür sehr viel des Equalizers hinzugefügt werden müsste, was die Verständlichkeit nicht fördern würde. Der GameSettingsChangedEvent dient als Beispiel für alle CustomEvent's.
Als Startpunkt 0./20. des Sequenzdiagrammes wurde die frameFinish()-Function auf der Node gewählt. Zu diesem Zeitpunkt wurde bereits das vorangehende FrameData-Package verarbeitet.
Die erste Prüfung, Schritt 2, betrifft die GameSettings, ob Änderungen gemacht wurden. Änderungen durch den GameSettingsChangedEvent werden da nicht eingerechnet. Ist dies der Fall wird Schritt 5 sendEvent() ausgeführt. Die Schritte 3-5 werden für alle Events, welche das SceneSetup verwaltet, durchgeführt.
sendEvent() synchronisiert die Daten mit der Master-Node des Equalizer (Master-Server), damit es beim nächsten handleEvent() der Config-Klasse verarbeitet werden kann. Dies geschieht bereits auf dem Master-Server, von wo aus das FrameData für die neuen Änderungen zusammengestellt wird.
Die Auswertung geschieht in Schritt 10 frameStart() auf den Pipe-Objekten. Es ist möglich, dass es mehrere Pipes gibt pro Computer (pro Graphic-Port). Dies bedeutet, dass der Zugriff auf die UserEvents, wie GameSettingsChangedEvent, mit einem Mutex auf dem FrameData-Object geschützt werden müssen. Dies ist im Sequenzdiagramm nicht ersichtlich.
Die GameSettings werden im Schritt 17 aktualisiert.
Anschliessend beginnt der Zyklus von vorne.

4.7.6 Level und Menu

Eine unserer Anforderungen an unser Squash ist, dass es in mehreren verschiedene Levels mit unterschiedlichen Eigenschaften gespielt werden kann. Dazu haben wir uns entschieden, die Implementation so zu gestalten, dass ein neues Level erstellt werden kann, ohne etwas neu zu programmieren und ohne neu zu kompilieren. Um dieses Ziel zu erreichen, haben wir das Level-Design in das Format XML umgeschrieben. Dasselbe Prinzip liess sich ebenfalls auf die Menu‘s übertragen. Dies hat den Vorteil, dass es weniger Programmierarbeit für neue Levels gibt und alles standardisiert wird.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.3.2 Level-Design mit OpenSceneGraph
- 6.4.3 Rendering Basic
- 6.5.7 Collisiondetection
- 6.5.13 Audio
- 6.5.14 Menuführung
- 6.6.3 VSP2: Verschiedene Räume

4.7.6.1 XML Schemas

Wir haben uns für das Format XML entschieden, weil das Format durch ein XML-Schema (XSD) definiert werden kann. Zudem können XML-Dateien mit einem üblichen XML-Editor gegen ein XML-Schema validiert werden kann. Durch ein XML-Schema kann sichergestellt werden, dass die einzelnen XML-Dateien gültig sind.

4.7.6.2 Generelle Knoten im XML

In jeder XML-Datei, Level und Menu, müssen am Anfang der Datei der Name und die Beschreibung gesetzt werden. Beim Level sind diese nur im XML definiert und werden nicht weiter angezeigt. Im Menu wird der im XML definierte Name jedoch als Titel dargestellt. So weiss der Benutzer immer, in welchem Menu er sich gerade befindet.

Zusätzlich müssen verschiedene Pfade angegeben werden:

- modelPath:
  Alle Modelle, die in dieser XML angegeben werden, müssen relativ zu diesem Pfad angegeben werden. Modelle werden nur im Knoten osg/model geladen.
- imagePath:
  Alle Bilddateien werden relativ zu diesem Pfad gesucht. Bilder werden nur bei den Texturen im Knoten osg/action/.
- soundPath:
  Alle Sounddateien werden relativ zu diesem Pfad gesucht. Sounds kommen hauptsächlich bei der Physik vor, um eine Kollision mit Sound zu unterstreichen. Dies betrifft den Knoten physics/actions/soundFIle.

- skyPath:
  Die Skybox hilft visuell, die Welt ansrpuchsvoller zu machen. Dazu werden die Bilder in dem hier angegebenen Pfad gesucht. Die Skybox wird in einem anderen Kapitel noch genauer beschrieben (siehe 6.2 Squash-Halle). Es braucht dazu die folgenden 6 Bilder im Format png:
  - down.png
  - east.png
  - north.png
  - south.png
  - up.png
  - west

4.7.6.3 Abbildung der Frameworks im XML

Der Grundgedanke war, alle Eigenschaften der Objekte für die verschiedenen Frameworks im XML definieren zu können. Folgende Frameworks galt es zu beachten:
- OSG (Darstellung)
- Bullet (Physik)
- Chai3D (Haptic)

Um eine klare Übersicht in der XML-Definition zu erhalten, haben wir alle Eigenschaften pro Framework gruppiert. Alle darstellungsspezifischen Werte sind unterhalb des XML-Knoten "osg" definiert. Alle physikalischen Eigenschaften für Bullet sind im XML-Knoten "physic" zu finden und alle Definitionen für die Haptic unterhalb von "haptics". Mit dieser Struktur kann die Übersicht gewahrt werden und die einzelnen Werte sind schnell gefunden.

OSG
Für das Framework OSG sind alle Informationen zur Darstellung wichtig. In diesem Teil eines Objektes kann die Farbe, die Textur sowie das Modell bestimmt werden.

Bullet
Für Bullet werden die Eigenschaften für die Restitution (Elastizität des Körpers) und der Friction (Reibung des Körpers) angegeben. In dieser Rubrik werden ebenfalls alle Aktionen definiert, die bei Kollisionen mit anderen Objekten ausgeführt werden sollen. Für ein Objekt können mehrere Aktionen für unterschiedliche Kollisions-Gruppen definiert werden. Werden für eine KollisionsGruppe mehrere Aktionen definiert, überschreibt die Letzte die vorherigen.

Chai3D
Unterhalb des Knotens "haptics" kann der Effekt für das Haptic-Gerät definiert werden, wenn dieses dem Objekt nahe kommt oder dieses berührt. Die bereits in I3D implementierten Effekte sind hier aufgeführt.

Die einzelnen Bereiche der Squash-Hallen sind in diesem XML-Schema vordefiniert worden. Ein einzelner Bereich wird durch ein Objekt dargestellt, welches im XML mit dem OsgObject abgebildet wird. In folgender Abbildung ist gezeigt, mit welchen Elementen im XML die verschiedenen Court-Elemente definiert wurden. Die Elemente enthalten die oben beschriebenen Eigenschaften:

```
<xs:complexType name="sideWall">
  <xs:sequence>
    <xs:element name="out" type="osgObject"></xs:element>
    <xs:element name="in" type="osgObject"></xs:element>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="osgObject">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="osg" type="osg"/>
    <xs:element name="physic" type="physic"/>
    <xs:element name="haptic" type="haptic"/>
  </xs:sequence>
</xs:complexType>
```

Abbildung 24: Definition im XML-Schema eines OsgObjects

Dies ist nur ein Auszug der Datei levelSchema.xsd.

Wie ein Objekt aus dem Squash-Court definiert wird, wird anhand folgenden Beispiels gezeigt: der Boden des ersten Level ist wie folgt definiert:

```
  <floor>
    <name>floor</name>
    <osg>
      <texture scaleFactorX="10" scaleFactorY="10">parquet.jpg</texture>
      <color r="1.0" g="0.0" b="0.5" a="1.0" />
    </osg>
    <physic>
      <collisionType>COL_FLOOR</collisionType>
      <restitution>0.9</restitution>
      <friction>1.5</friction>
      <action>
        <occuresWith>COL_BALL</occuresWith>
        <type>COLLISION_ACTION</type>
        <soundFile>explosion_2.mp3</soundFile>
      </action>
    </physic>
    <haptic>
      <effect>SURFACE_EFFECT</effect>
    </haptic>
  </floor>
```

Abbildung 25: Definition des Bodens im XML

Dies ist nur ein Auszug der Datei levels/1.xml.

- Der OSG-Knoten wird mit der Bilddatei 'parquet.jpg' texturiert mit einer Skalierung von 10 auf der X und Y- Achse. Zusätzlich wurde noch eine Farbe mitgegeben. Diese wird ignoriert sobald eine Textur angegeben wurde. Unter dem OSG Element kann zusätzlich ein Knoten

"trancparency" hinzugefügt werden. Mit dieser Fliesskomazahl von 0 bis 1 wird festgelegt, wie transparent der entsprechende OSG-Knoten sein soll.

- Der Boden wird der Gruppe COL_FLOOR zugewiesen. Somit wird dieser bei den Kollisionen als Boden erkannt.
- Die Elastizität beträgt für den Boden 0.9 und die Reibung ist bei 1.5. Diese Werte werden von Bullet verwendet. Die Elastizität ist ein Wert zwischen 0 und 1. Die Reibung kann frei gesetzt werden.
- Der Boden hat zudem eine Aktion gespeichert:
  - Die Aktion ist mit der Kollisionsgruppe COL_BALL verlinkt. Sobald der Ball mit dem Boden kollidiert, wird also diese Aktion ausgeführt.
  - Die Aktion ist vom Typ COLLISION_ACTION. Diese wird vom LevelCreator in die Aktion CollisionAction interpretiert.
  - Zu der Aktion wird noch eine Sounddatei angehängt, welche automatisch bei der Kollision mit dem Ball abgespielt wird.
- Als haptischen Effekt mit dem 3D Pointer der Haptic wird der SurfaceEffekt angewendet.

Der Boden (definiert in Abbildung 25: Definition des Bodens im XML) steht, sieht im Level so aus:

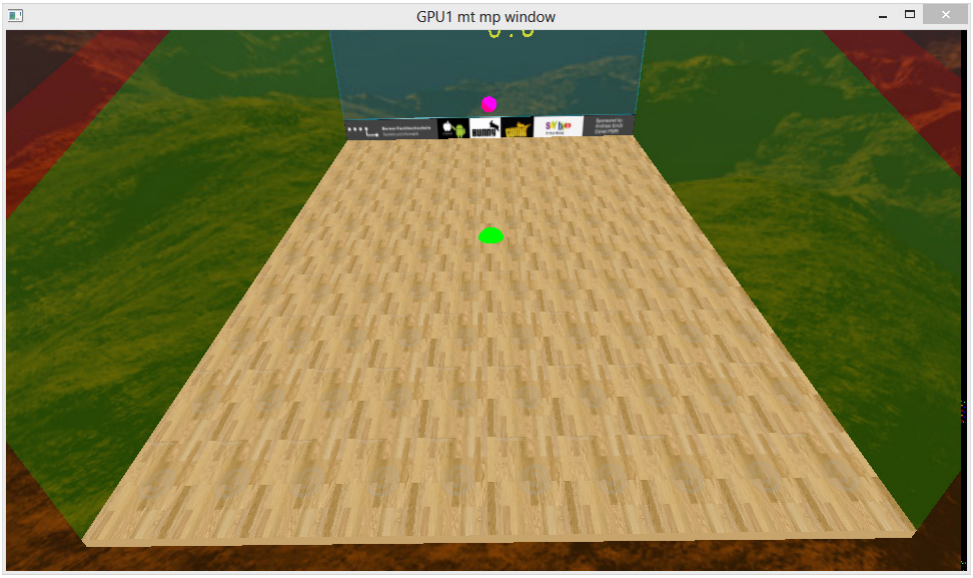


Abbildung 26: Der Boden hat die Textur parquet.jpg hinterlegt.

Falls der Boden in einer Farbe dargestellt werden soll, kann das Texture-Tag weggelassen werden (und gegebenenfalls eine Farbe hinzugefügt werden). Das obige Beispiel ohne Texture-Tag wird wie folgt dargestellt:

Abbildung 27: Der Boden hat die Farbe 1/0/0.5/1 (R/G/B/A) definiert.

4.7.6.4 Besonderheiten der Levels

Die Besonderheit der Level-XML-Definition ist, dass der Aufbau fix ist. Damit ist gemeint, ein Squash-Court besteht immer aus denselben Elementen: Einem Boden, zwei unterteilter Seitenwände, usw.
Jedes Level benötigt diese fix definierten Elemente. Das XML-Schema schreibt vor, dass jeder dieser Bereiche einzeln definiert werden muss. Die einzelnen Elemente werden wie im Beispiel in Kapitel "4.7.6.3 Abbildung der Frameworks im XML" definiert. Mit dieser Definition ist es möglich innert kürzester Zeit einen neuen Squash-Raum, mit eigenen physikalischen Eigenschaften, zu definieren.

4.7.6.5 Besonderheiten der Menu‘s

Während das Level von der Struktur her immer gleich aufgebaut ist, sieht praktisch jedes Menu anders aus. Jedes Menu enthält andere und unterschiedlich viele Menu-Punkte. Somit muss das XML-Schema eine Struktur aufweisen, welche erlaubt, unterschiedlich viele Menu-Punkte zu deklarieren und diese entsprechend zu verwalten. Statt der fix definierten XML-Elemente "OsgObject" werden dynamische MenuEntry-Elemente verwendet. Diese Elemente können beliebig oft vorkommen. Der MenuCreator muss diese XML-Elemente als Liste verarbeiten. Jeder Menu-Punkt kann mit einer Aktion MenuAction oder MenuGameSettingsAction versehen werden.

Hier kurz die Unterschiede zwischen den beiden Actions (beschrieben unter 4.7.9.2 Aktionen auf den Objekten):
MenuAction: Mit dieser Aktion kann ein neues Menu oder das momentan gewählte Level in den SceneManager geladen werden. Es dient also zur Navigation durch die verschiedenen Menus und das Spiel.
MenuGameSettingsAction: Diese Aktion wird verwendet, um die Einstellungen der GameSettings auf dem SceneSetup zu verändern.

Die einzelne XML-Definition des Menu-Punktes (entry) ist ähnlich der XML-Definition des OsgObject‘s aus dem Level. Jedoch lässt die Definition eine Liste der entry-Elemente zu. Die Reihenfolge der Menu-Punkte, wie sie visuell dargestellt werden, hängt von der Reihenfolge im XML ab.

```
<xs:complexType name="entries">
  <xs:sequence>
    <xs:element name="entry" type="entry" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>


<xs:complexType name="entry">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="osg" type="osg"/>
    <xs:element name="physic" type="physic"/>
    <xs:element name="haptic" type="haptic"/>
  </xs:sequence>
</xs:complexType>
```

Abbildung 28: Definition im XML-Schema des Menu’s

Dies ist nur ein Auszug der Datei menuSchema.xsd.

Durch diese Struktur ist es möglich, ein Menu mit nur einem oder unterschiedlich vielen Einträgen zu erstellen. Jeder davon wird als Eintrag interpretiert und mit den entsprechenden Werten initialisiert, um das Menu aufzubauen.

Im Hauptmenu kann mit dem Menu-Punkt "Einstellungen" in das Einstellungen-Menu gewechselt werden. Dieses Einstellungsmenu hat die Id 3. Im XML vom Hauptmenu wird dieser Punkt wie folgt definiert:

```
<entry>
  <name>Einstellungen</name>
  <osg>
    <color r="0.13" g="0.13" b="0.13" a="1.0"/>
    <colorSelected r="1.0" g="0.5" b="0.0" a="1.0"/>
  </osg>
  <physic>
    <restitution>0.0</restitution>
    <friction>0.0</friction>
    <event>
      <type>LOAD_MENU</type>
      <menuId>3</menuId>
    </event>
  </physic>
  <haptic>
    <effect>SURFACE_EFFECT</effect>
  </haptic>
</entry>
```

Abbildung 29: XML für ein Menu-Punkt

- Der Menu-Punkt wird mit "Einstellungen" beschriftet. Dieser darf unterschiedlich dem Namen des Menu's sein, welches geladen wird und ist frei wählbar.
- Jeder Menu-Punkt hat zwei Farben:
  - **Color:** Die normale Farbe des Menu-Punktes. Diese ist dem OSG-Knoten dann gesetzt, wenn der Menu-Punkt momentan nicht aktiv ist.
  - **ColorSelected:** Ist der Menu-Punkt momentan ausgewählt, so wird dieser in der definierten Farbe dargestellt.
- Die Physik-Eigenschaften spielen hier keine Rolle und sind deswegen auf 0 gesetzt.
- Für einen Menu-Punkt muss genau eine Aktion definiert werden. Die wird im MenuCreator automatisch mit der Kollisionsgruppe COL_MENU_ENTRY gesetzt. Diese Aktion kann eine der folgenden sein:
  - MenuAction
    Sie hat entweder das Attribut LOAD_MENU: das Menu mit der Id definiert im menuId-Tag wird geladen.
    Oder diese hat das Attribut LOAD_LEVEL: das Menu mit der Id definiert im levelId-Tag wird geladen oder falls dieses nicht gesetzt ist wird die Id von den GameSettings genommen.
  - MenuGameSettingsAction:
    Der Typ CHANGE_GAME_SETTINGS verändert die Eigenschaften und benötigt zusätzlich das XML-Element:
    - settingProperty: Bestimmt die Eigenschaft der Settings die verändert werden sollen.
    - changeByValue: Wird dieser Wert gesetzt, verändert sich der Wert der GameSettings relativ um diesen Wert.
    - setValue: Wird dieser Wert gesetzt, wird der Wert der GameSettings auf diesen Wert gesetzt.
  - ExitAction
    Der Typ EXIT beendet das Spiel.

- Das Haptic-Element definiert den Effekt, welcher beim Berühren mit dem Haptic-Gerät verwendet werden soll.

Das Menu mit dem "Einstellungen"-Punkt, wie oben beschrieben, sieht im SceneManager wie folgt aus:

Abbildung 30: Beispiel Menu-Punkt "Einstellungen".

Wird die Farbe ColorSelected auf Rot (r = 1.0, g = 0, b = 0, a = 1) gesetzt, nimmt der Punkt Einstellungen die rote Farbe an, sobald dieser angewählt ist. Dies ist in nachfolgendem Bild zu sehen:

Abbildung 31: Menu-Punkt mit veränderter Farbe

4.7.6.6 Menuführung

Auf die Menuführung aus dem Pflichtenheft (6.5.14 Menuführung) wird hier kurz gesondert eingegangen, um weitere spezielle Gegebenheiten des Menus aufzuzeigen.

Die Tasten-Funktionen sind unter Kapitel 7.3 Tastenkombination. Die Navigation mit Hilfe des haptischen Gerätes ist im Kapitel 7.4 Haptic-Steuerung.

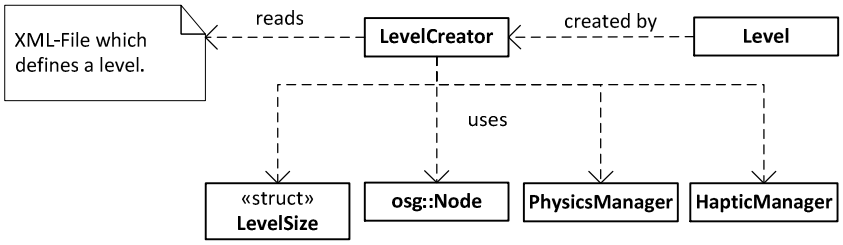
Die Auswahl eines Menu-Punktes geschieht über die Kollision zwischen Racket-Objekt und dem Menu-Punkt-Objekt, ausgelöst durch Bullet.

Bei der Haptic ist wichtig, dass nur der erste Spieler (roter Schläger) durch das Menu navigieren darf. Die Kollisionen mit dem zweiten Spieler werden ignoriert. Somit kann einer bestimmen, was wie eingestellt wird, ohne durch den anderen Spieler gestört zu werden.

Abbildung 32: Zwei Spieler im Menu mit Schatten.

4.7.6.7 Implementierung

Bei der Implementierung war zuerst ein Builder-Pattern angedacht. Dies haben wir verworfen und uns für ein Konzept mit einer XML-basierten Definition entschlossen. Mit Hilfe dieses Konzepts kann für ein neues Menu/ Level eine XML-Datei definiert werden. Das Interpretieren dieser Datei geschieht in einer Creator-Klasse. Dabei wird eine Creator-Klasse für das Level und eine für das Menu definiert. Beide Klassen interpretieren die jeweiligen XML-Dateien und erstellen so die gewünschte Szene. In folgendem Diagramm sind die Abhängigkeiten dieser Klassen zu den nächsten wichtigen Klassen aufgezeigt.



- Der LevelCreator öffnet die angegebene Level-XML-Datei.
- Kann die Datei gelesen werden, erstellt der LevelCreator eine neue Instanz des Level‘s.
- Für jedes Objekt des Squash-Courts
  - wird das entsprechende XML-Element gesucht und gelesen.
  - werden die Werte aus dem XML an OST, Bullet und die Haptic übergeben
- Zu dem Level wird noch die Skybox hinzugefügt.
- Als letztes wird dem Level das gesamte Modell der Squash-Halle hinzugefügt.
- Das Level wird an den SceneManager übergeben, welcher dieses korrekt lädt und darstellt.

Analog dazu wird ein MenuManager durch den MenuCreator erstellt. Im Unterschied zum LevelCreator interpretiert der MenuCreator die Anzahl Menu-Punkte aus dem XML und fügt diese dem MenuManager hinzu.

Die Gesamtheit der Klassen um ein Level oder ein Menu zu generieren sind die folgenden im UML-Diagramm:

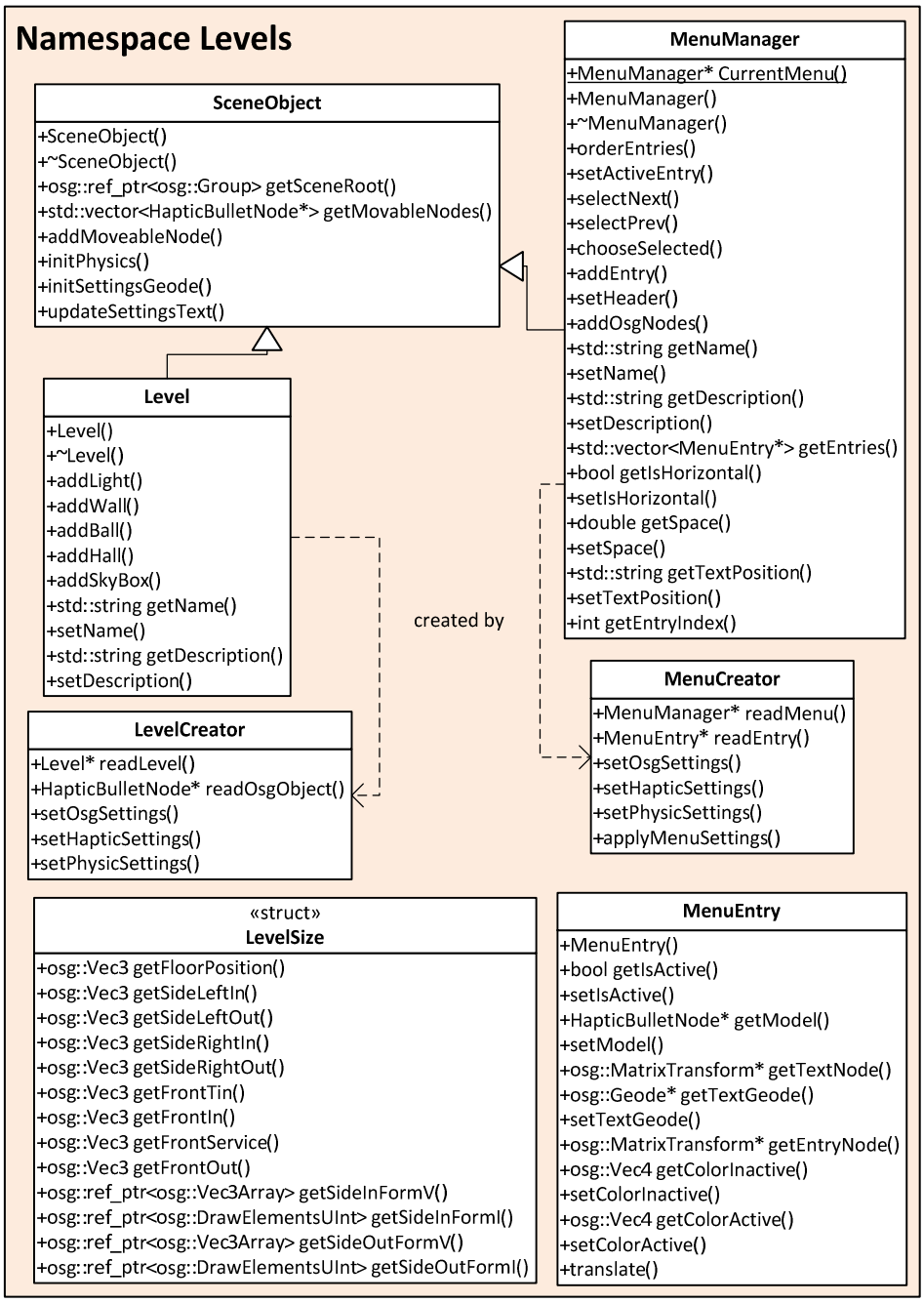


Abbildung 33: UML-Diagramm der Level- und Menuerstellung

Einige Klassen wurden bereits erwähnt, hier noch einmal alle im Überblick:

SceneObject:
Das SceneObject ist eine Abstraktion für jedes Objekt, das vom SceneManager geladen, verwaltet und dargestellt werden kann. Es werden einige Funktionalitäten definiert, die im Zusammenhang mit dem OSG laden und löschen stehen.

Level:
Das Level ist eine Ableitung des SceneObject's. Es stellt ein Level mit einem Squash-Court in dem Spiel dar. Der LevelCreator erstellt und initialisiert das Level mit den nötigen Objekten und Werte.

LevelCreator:
Der LevelCreator liest die korrekte XML-Datei und erstellt daraus ein gültiges Level-Objekt, welches dem SceneManager übergeben werden kann.

LevelSize:
Die LevelSize ist eine Struktur mit Konstanten der von uns definierten Werten. Sie beinhaltet alle wichtigen Dimensionen, um das Squash-Court dementsprechend aufzubauen. Die Verhältnisse wurden einem echten Squash-Court nachempfunden. Dazu haben wir die Werte von der Wikipedia-Seite[1] übernommen.

MenuManager:
Der MenuManager erbt wie auch das Level vom SceneObject und kann im SceneManager verwaltet werden. Zusätzlich werden auf dieser Klasse weitere Funktionalitäten zur Verfügung gestellt, damit zum Beispiel durch das Menu navigiert werden kann. Ebenfalls kann durch den MenuManager ein neuer Menu-Punkt angewählt werden (sei dies durch die Hapitc oder durch Tastaturbefehle). Aus diesem Grund haben wir es nicht nur Menu, sondern MenuManager benannt. Es stellt zwar ein Menu selber dar, hat aber zusätzliche Funktionen.

MenuCreator:
Der MenuCreator liest die korrekte XML-Datei und erstellt daraus ein gültiges MenuManager-Objekt, welches dem SceneManager übergeben werden kann.

MenuEntry:
Das Menu besteht aus mehreren verschiedenen Menu-Punkten. Ein MenuEntry stellt ein solcher Menu-Punkt dar. Dieser beinhaltet und verwaltet das OSG Modell, welches mit Bullet und der Haptic verknüpft ist, sowie die entsprechende Beschriftung des Menu-Punktes.

[1] http://de.wikipedia.org/w/index.php?title=Datei:Squash_Court_German.jpg&filetimestamp=20081023112321

4.7.7 GamePlay - BallStateMachine

Die Squash-Regeln bildeten wir mit Hilfe des State Pattern als BallStateMachine ab. Mit Hilfe der BallStateMachine wird der gesamte Spielablauf auf seine Gültigkeit überprüft. Die Regeln sind leicht angepasst. Als Bespiel beachten wir nicht die komplette Aufschlagsregel, sondern wenden eine vereinfachte an.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.5.1 Spiel Multiplayer
- 6.5.2 Spiel Singleplayer

4.7.7.1 Die Regeln

Wie im Pflichtenheft bereits beschrieben, haben wir folgende Regeln für unser Squash definiert:

- Der Ball muss nach jedem Schlag auf direktem oder indirektem Weg die Vorderwand berühren. Als indirekt gilt ein Weg über die Seitenwände.
- Der Ball darf auf dem Weg von der Vorderwand zum Spieler nicht mehr als einmal den Boden berühren.
- Ein Ball gilt als im „Aus“, wenn er die Wände oberhalb dort angebrachter roter Begrenzungslinien, die Begrenzungslinie selbst oder das Tin berührt.
- Die Rückwand gilt als „Aus“.

Diese Regeln haben wir während der Umsetzung mit folgenden Regeln ergänzt:

- Der Ball muss beim Aufschlag an der Vorderwand über der definierten Aufschlagslinie aufschlagen. Ansonsten gibt es einen Punktgewinn für den Gegner.

4.7.7.2 Kollisionen für die Zustandsänderung

Ein Zustand stellt eine Spiel-Situation dar. Beispiel: Aufschlag Spieler 1.
Die Zustandsübergänge werden durch Kollisionen, ausgelöst durch die Physik-Engine Bullet, gekennzeichnet.
Da nur Kollisionen zwischen dem Ball und einem anderen Objekt relevant sind, haben wir diese Klassen speziell für die BallStateMachine definiert. Sobald eine Kollision mit einem Objekt dieser Klassen auftritt, reagiert die BallStateMachine darauf. Die Kollision zwischen dem Ball und dem Boden wird zum Beispiel so gewertet, dass die BallStateMachine versucht, den neuen Status "Floor" zu setzen. Ist dieser Statuswechsel gültig, wird der neue Status gesetzt. Ansonsten wird ein Event geschickt, so dass die BallStatemachine ungültig und die Punkte der Spieler entsprechend angepasst wird.

Um bei den Kollisionen den daraus folgenden Status herauszufinden, mussten die verschiedenen Objekte mit einer neuen Eigenschaft erweitert werden. Diese enthält den nächsten Status für die BallStateMachine. Damit die möglichen Werte dieses neuen Attributes festgelegt werden konnten, ist der Squash-Court in einzelnen wichtige Bereiche aufgeteilt. Wichtige Bereich deshalb, da es auch Bereich ausserhalb des Courts gibt, die keinen Einfluss haben sollen. Die einzelnen Bereiche definieren dann die verschiedenen Zustände der BallStateMachine. So ist es möglich, anhand der Kollision schnell den nächsten Status festzulegen. Mit Hilfe dieser Aufteilung konnten wir eine effiziente BallStateMachine definieren. Die verschiedenen Transitions lassen sich sehr schnell daraus ablesen.

Auf der folgenden Grafik sind die verschiedenen Bereiche farblich dargestellt und in der darauf folgenden Tabelle näher erläutert. In der Tabelle ist näher erläutert, welche Teile des Squash-Courts in einen einzelnen Bereich fallen.

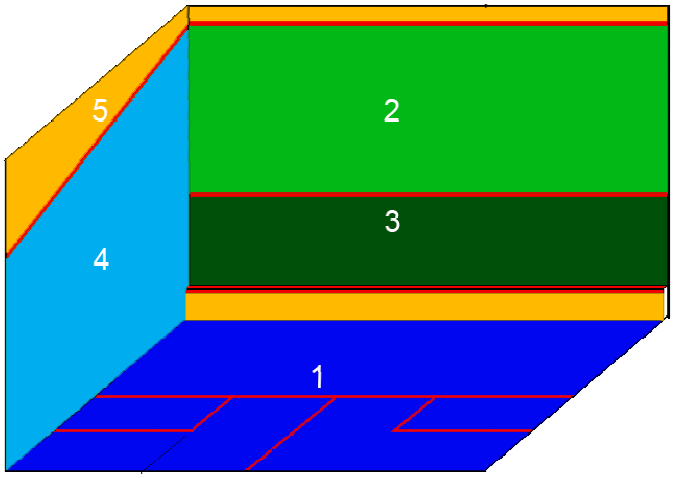


Abbildung 34: Der Squash-Court, aufgeteilt in die einzelnen Bereiche.

	Ball-Status	**Betrifft**	**Beschreibung**
1	Floor	Floor	Der gesamte Boden gibt diesen Status zurück. Eine Unterteilung der einzelnen Aufschlagsfeldern wurde der Einfachheit halber nicht umgesetzt. Die BallStateMachine könnte aber entsprechend angepasst werden, damit auch noch die Zusatzregel beim Anspiel berücksichtig wird.
2	FrontService	FrontService	Beim Anschlag muss der Ball auf diese Fläche prallen. Diese Unterteilung dient dazu, folgende Regel umzusetzen: Beim Anspiel muss der Ball zwingend über die Aufschlaglinie (zwischen 2 und 3) treffen.
3	FrontIn	FrontIn	Der Ball muss bei jedem Ballwechsel auf diese oder die FrontService treffen.
4	SideIn	LeftSideIn, RightSideIn, BackSideIn	Die Seitenwand ist durch eine Outlinie unterteilt. Der Ball muss beim Auftreffen einer solchen Wand zwingend unterhalb dieser Linie sein.
5	Out	LeftSideOut, RightSideOut, BackSideOut, FrontOut, FrontTin	All diese Bereiche sind definiert, dass der Ball im Aus ist. Mit diesem wird die BallStateMachine in einen ungültigen Status versetzt, womit diese reagieren kann.

6	Racket	Racket 1 & 2	Die verschiedenen Rackets der beiden Spieler sind für einen erfolgreichen Ballwechsel nötig.

Tabelle 11: Beschreibung der Bereiche im Squash-Court

4.7.7.3 Erlaubte Zustandsübergänge (Transition)

Anhand der Regeln lässt sich nur herauslesen, welche Zustandsübergänge erlaubt sind. Um dies genauer aufzuzeigen, erklären wir hier kurz, wie wir die erste Regel umgesetzt haben.

Zur Erinnerung, die Regel lautet:
- Der Ball muss nach jedem Schlag auf direktem oder indirektem Weg die Vorderwand berühren. Als indirekt gilt ein Weg über Seitenwand.

Schlussfolgerungen:
- Jeder Schlag durch den Schäger wird durch den State Racket repräsentiert.
- Sobald der Ball vom Schläger gespielt wird, darf dieser nur noch die Bereiche SideIn berühren, um schlussendlich an dem FrontIn oder FrontService abzuprallen.
- Vom Schläger zur Vorderwand darf der Ball beliebig oft an einer Seitenwand abprallen.

Bei dieser Regel wird also ersichtlich, dass vom Zustand Racket folgende Zustände folgen können:
- SideIn
- FrontIn
- ServiceIn

Aus diesen Regeln lassen sich alle erlaubten Zustandsübergänge in einer Tabelle definieren. Falls notwendig sind ebenfalls die zusätzlichen Überprüfungen definiert. Im Folgenden ein kleiner Auszug aus dieser Tabelle:

Von Zustand	**Ziel Zustand**	**Zusätzliche Überprüfung**
Start	Racket	
Racket	FrontIn	Der Ball darf nicht im Aufschlagsmodus sein.
Racket	FrontService	
Racket	SideIn	
SideIn	Floor	Der Boden darf noch nie berührt worden sein.
SideIn	FrontIn	Der Ball darf nicht im Aufschlagsmodus sein.
SideIn	FrontService	
SideIn	Racket	Der Ball muss bereits an der Vorderwand

		aufgeprallt sein. Zusätzlich im Multiplayer: Nach Spieler 1 kommt Spieler 2
...	..	..

Tabelle 12: Erlaubte Zustandsübergänge (Auszug)

Weil beim Aufschlag der Ball zwingend oberhalb der Aufschlaglinie sein muss, hat dem Zustandsübergang vom Racket zum FrontIn noch eine zusätzliche Prüfung zu erfolgen. Diese muss prüfen, ob der Ball im Aufschlag-Modus ist, falls ja, ist dieser Zustandsübergang ungültig. Dies kann man aus der obigen Tabelle aus der Spalte Racket - FrontIn entnehmen.

4.7.7.4 TransitionChecks

Um zum Beispiel den Fall zu überprüfen, welcher im vorherigen Kapitel beschrieben wurde, haben wir die TransitionChecks eingeführt. Ein TransitionCheck kann zusätzlich an die Transition angehängt werden. Sobald die Transition einen solchen Check besitzt, muss dieser ebenfalls erfolgreich durchlaufen sein, bevor der Zustandsübergang erlaubt wird.

In dem Beispiel vom Racket zu FrontIn überprüft also der TransitionCheck, in welchem Modus der Ball ist. Ist dieser im Aufschlagsmodus, wird der Test negativ ausfallen, ansonsten positiv.

4.7.7.5 TransitionManager

Um die Zustandsübergänge zu überwachen, wird ein sogenannter TransitionManager eingesetzt. Dieser verwaltet alle erlaubten Zustandsänderungen. Um der BallStateMachine den Übergang vom Schläger (Racket1) zu der Vorderwand (FrontIn) zu erlauben, muss dem TransitionManager eine neue Transition von Racket zu FrontIn hinzugefügt werden. Zusätzlich muss dieser Transition noch einen Check angehängt werden, welcher die Gültigkeit des Aufschlages prüft (wie im vorherigen Kapitel beschrieben).

Sobald die BallStateMachine versucht vom Status Racket in einen neuen Status überzugehen, wird dies im TransistionManager auf eine erlaubte Transition geprüft. Ist der Übergang erlaubt, wechselt die BallStateMachine in den neuen Status und verändert die State-Variablen entsprechend. Ansonsten wird der Übergang nicht erlaubt und die BallStateMachine muss diesen Versuch als Ungültig behandeln.

4.7.7.6 Implementation

Die Implementation der BallStateMachine für den Ball, verwendet ein übliches State-Pattern. Zusätzlich werden die Zustandsübergänge mit einem TransitionManager, welcher die gültigen Transitions verwaltet, überprüft.

Jeder Bereich im Squash-Court kriegt seinen eigenen State. So kann bei der Kollision schnell der entsprechende neue State gefunden werden. Dieser wird dann an die BallStateMachine übergeben. Bevor diese den neuen Status akzeptiert, wird im TransitionManager auf eine gültige Transition geprüft. Falls eine ohne zusätzlichen TransitionCheck vorhanden ist, wird der Übergang erlaubt. Ist ein zusätzlicher TransitionCheck angehängt, wird entsprechend dessen Gültigkeit der Übergang erlaubt oder verweigert. Wenn der Übergang verweigert wird, wird der Game-State entsprechend verändert. Je nachdem welcher Spieler den Fehler begangen hat, bekommt der entsprechende Spieler einen Punkt. Ist der Übergang erlaubt, wird dieser übernommen und die BallStateMachine wird entsprechend der execute-Methode des neuen Status aktualisiert.

Im folgenden UML-Diagramm wird ersichtlich, wie die verschiedenen Klassen zusammenspielen.

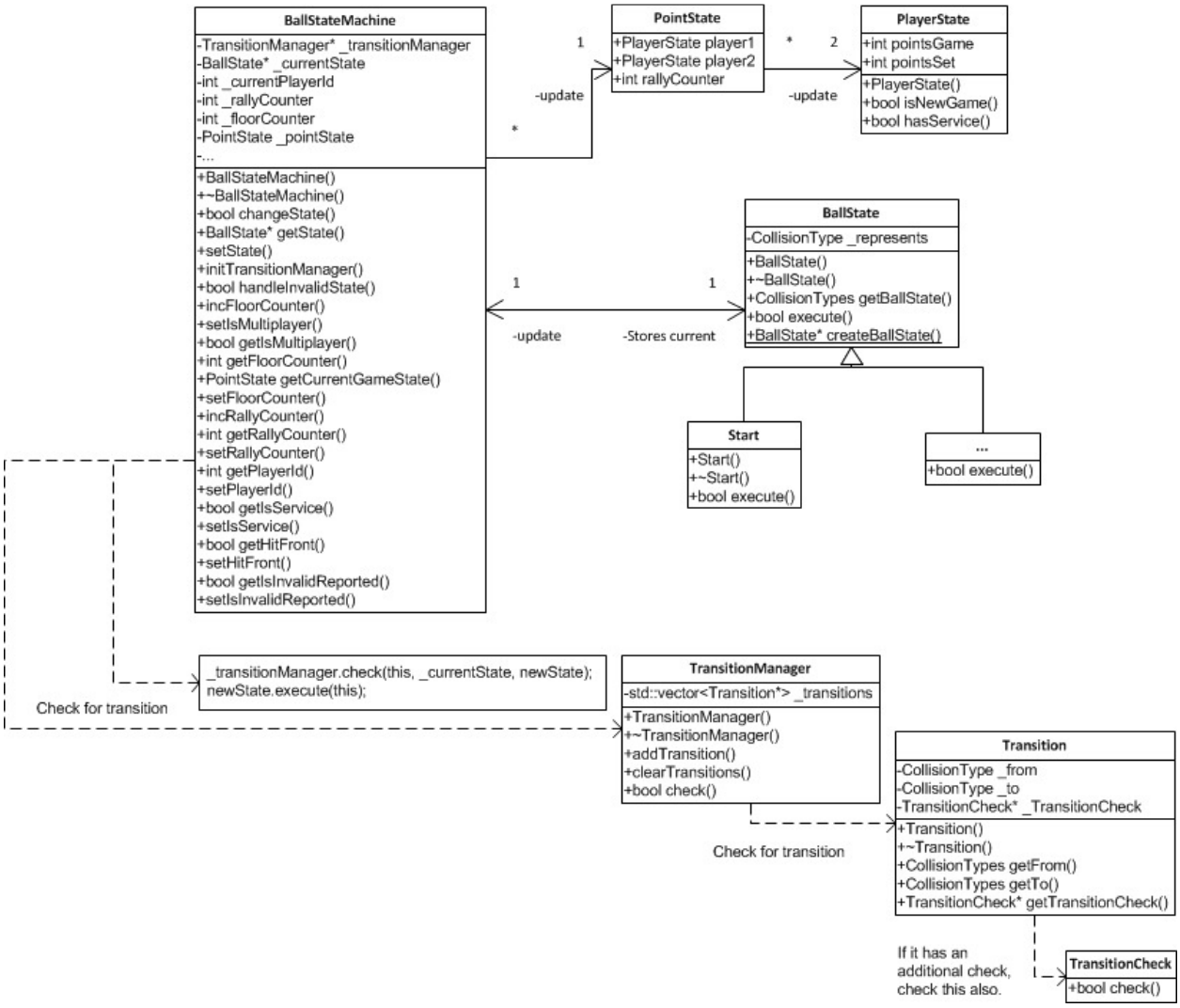


Tabelle 13: Klassendiagramm BallStateMachine

Werden die üblichen UML-Diagramme für das State-Pattern mit unserem verglichen, fällt auf, dass meistens nur die obere Hälfte (ohne PointState, PlayerState, TransitionManager, Transition) dazu gehört. Beim Ausarbeiten des Konzeptes für dies BallStateMachine haben wir uns darauf geeinigt das übliche State-Pattern um eine Transition-Verwaltung zu erweitern. Mit der Hilfe eines TransitionManager‘s, in welchem Transitions von einem Von-Zustand zu einem Ziel-Zustand definiert sind, haben wir eine einfache Lösung gefunden, die unsere Anforderungen abdeckt. Die BallStateMachine selber initialisiert die erlaubten Transitions, welche zugelassen werden sollen. So kann auf der BallStateMachine umgestellt werden, ob die Regeln für einen oder zwei Spieler gelten sollen. Je nachdem werden andere Transitions mit anderen zusätzlichen Checks initialisiert und der mögliche Spielablauf beeinflusst.

Ein weiterer Vorteil mit dieser Implementierung ist, dass wir uns die Möglichkeit gelassen haben, die erlaubten Zustandsübergänge ebenfalls dynamisch zur Laufzeit anzupassen. Wir können das Spiel also soweit erweitern, dass ab einer gewissen Anzahl Punkte plötzlich ein Spieler nur noch direkt an die Vorderwand schlagen muss (Seitenwand nicht mehr erlaubt,

Handicaps). Diese Möglichkeit haben wir aus Zeitknappheit nicht mehr nutzen können. Mit wenig Aufwand wäre dies jedoch realisierbar.

In dem folgenden UML Diagramm sind alle Klassen, die zu dieser BallStateMachine gehören. So erkennt man alle verschiedenen möglichen Status und alle nötigen zusätzlichen Überprüfungen für Zustandsänderungen für unser Squash.

Die Klassen PointState und PlayerState bilden den aktuellen Punktestand ab.

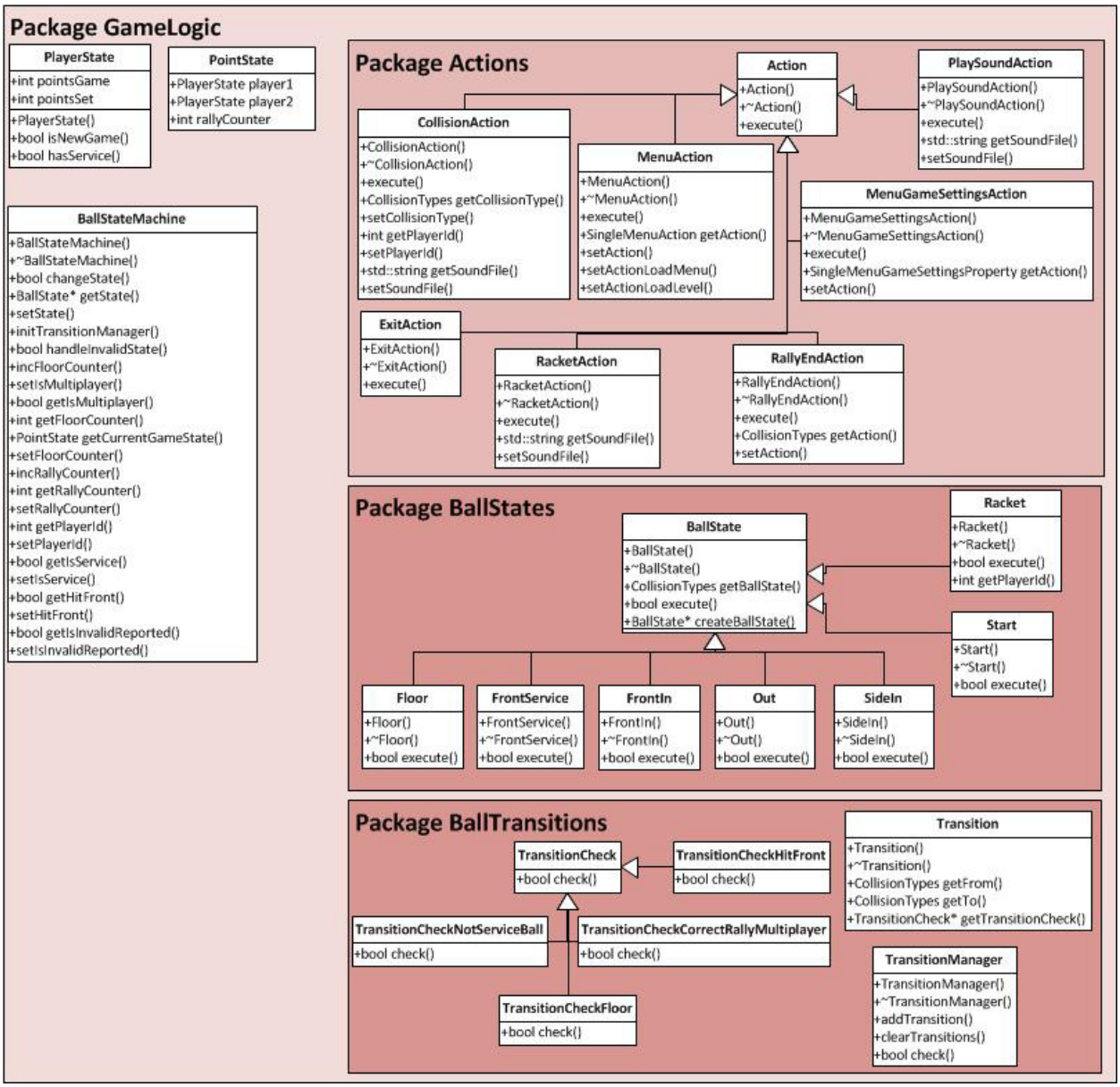


Abbildung 35: Klassen GameLogic

Beispiel einer Transition:

Um den Zustandsübergang vom Racket zur Seitenwand zu erlauben, muss dem TransitionManager folgende Transition hinzugefügt werden:

```
_transitionManager->addTransition(
      new BallTransitions::Transition(
            SQUASHI3D::Physics::CollisionHandler::COL_RACKET,
            SQUASHI3D::Physics::CollisionHandler::COL_SIDE_IN
      )
);
```

Wenn die BallStateMachine im Zustand Racket ist und eine Kollision zwischen dem Ball und einer Seitenwand auftritt, versucht die BallStateMachine in den Zustand SideIn zu wechseln. Dafür wird im TransitionManager auf eine gültige Transition geprüft, welche im obigen Beispiel gegeben ist. Da er diese findet und die keine zusätzliche Überprüfung benötigt, wird der Zustandswechsel erlaubt.

Ein weiteres Beispiel soll den Zustandsübergang vom Racket zu dem Bereich FrontIn aufzeigen. Bei diesem Übergang ist es wichtig, dass diese beim Aufschlag ebenfalls als Out gewertet wird, weil beim Aufschlag oberhalb der Aufschlaglinie gespielt werden muss. Somit braucht dieser Übergang eine zusätzliche Überprüfung. Dies wird in unserem System wie folgt vorgenommen:

```
_transitionManager->addTransition(
      new BallTransitions::Transition(
            SQUASHI3D::Physics::CollisionHandler::COL_RACKET,
            SQUASHI3D::Physics::CollisionHandler::COL_FRONT_IN,
            new BallTransitions::TransitionCheckNotServiceBall()
      )
);
```

Sobald vom Zustand Racket in den Zustand FrontIn gewechselt werden soll, findet der TransitionManager die zusätzliche Überprüfung TransitionCheckNotServiceBall. Diese überprüft lediglich, ob der Ball im Aufschlagmodus ist. Ist der Ball im Aufschlagmodus wird mit false der Zustandswechsel nicht erlaubt, ansonsten wird dieser mit true zugelassen.

4.7.8 Kollisionserkennung in Bullet

Eine Kollision geschieht, wenn zwei Objekte auf einander treffen. Die Physik-Engine Bullet stellt solche Berechnungen zur Verfügung. Mit den Funktionen von Bullet können Kollisionen, welche virtuell stattfinden, ausgelesen werden.
Um in der 3D Welt mit dem haptischen Gerät zu navigieren und ebenfalls um die BallStateMachine richtig zu aktualisieren, werden alle Kollisionen der Bullet-Welt (Physik-Engine) benötigt. Diese Kollisionen werden durch Bullet dem PhysicsManager gemeldet, welcher diese entsprechend aufbereitet. Jedes Kollisions-relevante Objekt muss eine CollisionSpecification besitzten, um vom CollisionManager korrekt verarbeitet werden zu können. Haben beide Kollisions-Objekte eine CollisionSpecification definiert, wird diese Kollision an den CollisionManager weitergeleitet. Andernfalls wird die Kollision ignoriert.

Rollte eine Kugel am Boden, so sendet Bullet andauernd Kollisionen zwischen Kugel und Boden. Für SquashI3D interessiert nur der Anfang einer Kollision und nicht die Dauer. Der Zeitpunkt wann die Kugel das erste Mal auf den Boden trifft, bis sie den Boden wieder verlässt, wird nachfolgend als Sequenz bezeichnet.
Deshalb prüft der CollisionManager, ob eine Kollision das erste Mal in einer Sequenz oder mitten in der Sequenz auftritt. Nur die erste Kollision einer Sequenz wird an alle anderen Computer via CollisionEvent weitergeleitet.

Implementierte Features (Kapitel gem. Pflichtenheft):

- 6.5.7 Collisondetection

4.7.8.1 Kollisionssequenz von Bullet

In Bullet werden die Kollisionen pro gerechneten Physik-Frame ermittelt. Die SquashI3D Kollisionserkennung ist als Callback-Funktion auf der Physik-Welt definiert. Diese wird nach jedem Update der Physik automatisch aufgerufen. In diesem Callback werden alle von Bullet gefundenen Kollisionen überprüft, ob diese die CollisionSpecification spezifiziert haben. Alle gültigen Kollisionen werden dann an den CollisionManager geschickt, welcher die aktuellen, alten und neuen Kollisionen erkennt und entsprechend aktualisiert, hinzufügt oder löscht. Auf diese Weise wird gewährleistet, dass die von Bullet mehrmals erkannte gleiche Kollision nicht bei jedem Durchlauf als neu erkannt wird.

Der Grund, weshalb ein solcher Trigger eingesetzt wird, liegt in der Funktionsweise von Bullet begründet. Bullet erkennt alle aktuellen Kollisionen und speichert nicht die Folge von gleichen Kollisionen ab. Eine solche Folge tritt immer dann auf, wenn eine Kollision länger als ein Update-Zyklus des PhysicManager‘s dauert. Dies ist zum Beispiel bei der Kollision zwischen dem Ball und dem Boden so, wenn der Ball auf dem Boden aufprallt, dieser sich durch die Kraft deformiert und somit länger auf dem Boden haftet. In der folgenden Tabelle wird der gesamte Vorgang einer Kollision in Bullet erläutert.

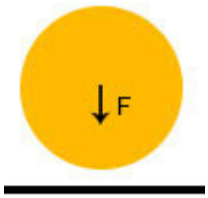
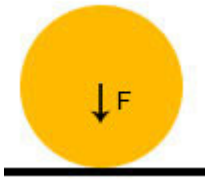
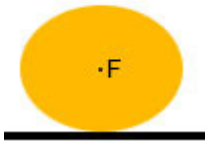
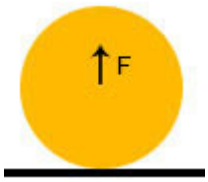
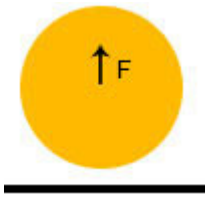
Zustand (Bild)	Kollisions-Nummer	Beschreibung
	-	Keine Kollision tritt auf
.. x_1 Frames verstreichen hier.		
	1	Der Ball berührt den Boden das erste Mal und löst somit die erste erkannte Kollision aus.
.. x_2 Frames verstreichen hier.		
	y	Der Ball hat sich bis jetzt maximal deformiert. Der Kollisionsablauf ist in der Mitte angelangt. Die Beschleunigungskraft wird hier umgekehrt.
.. x_3 Frames verstreichen hier.		
	z	Der Ball hat seine maximale Beschleunigung wieder erreicht und berührt den Boden noch das letzte Mal. Dieselbe Kollision ist nun schon z mal erkannt worden.
.. x_4 Frames verstreichen hier.		
		Es tritt keine Kollision mehr auf.

Tabelle 14: Sequenz einer Kollision in Bullet

Dieses Phänomen können wir nachstellen, indem wir die Ausgabe auf dem CollisionManager entsprechend einstellen. So sehen wir, dass die Kollision beim Aufprallen des Balles mit dem Boden in diesem Beispiel insgesamt 12 Frames dauerte:

2013-Jan-04 11:40:19.252493 <notification> Collision between: ball and floor (0)
2013-Jan-04 11:40:19.279494 <notification> Collision between: ball and floor (1)
2013-Jan-04 11:40:19.284494 <notification> Collision between: ball and floor (2)

2013-Jan-04 11:40:19.288495 <notification> Collision between: ball and floor (3)
2013-Jan-04 11:40:19.292495 <notification> Collision between: ball and floor (4)
2013-Jan-04 11:40:19.305496 <notification> Collision between: ball and floor (5)
....
2013-Jan-04 11:40:19.342498 <notification> Collision between: ball and floor (12)

Es wird deutlich, dass der CollisionManager die entsprechende Kollision als Abfolge erkannt und somit den Wert bei jedem Update um eines hochgezählt hat. Nur bei der ersten Kollision wird der CollisionEvent mit den Strings "ball" und "floor" an die anderen Clients (Nodes) geschickt. So erscheint der CollisionEvent auf allen Clients (Nodes) nur einmal. Auf diese Weise wird gewährleistet, dass die BallStateMachine korrekt aktualisiert werden kann, weil ein neuer Status nur einmal geschickt wird..

4.7.8.2 Implementation

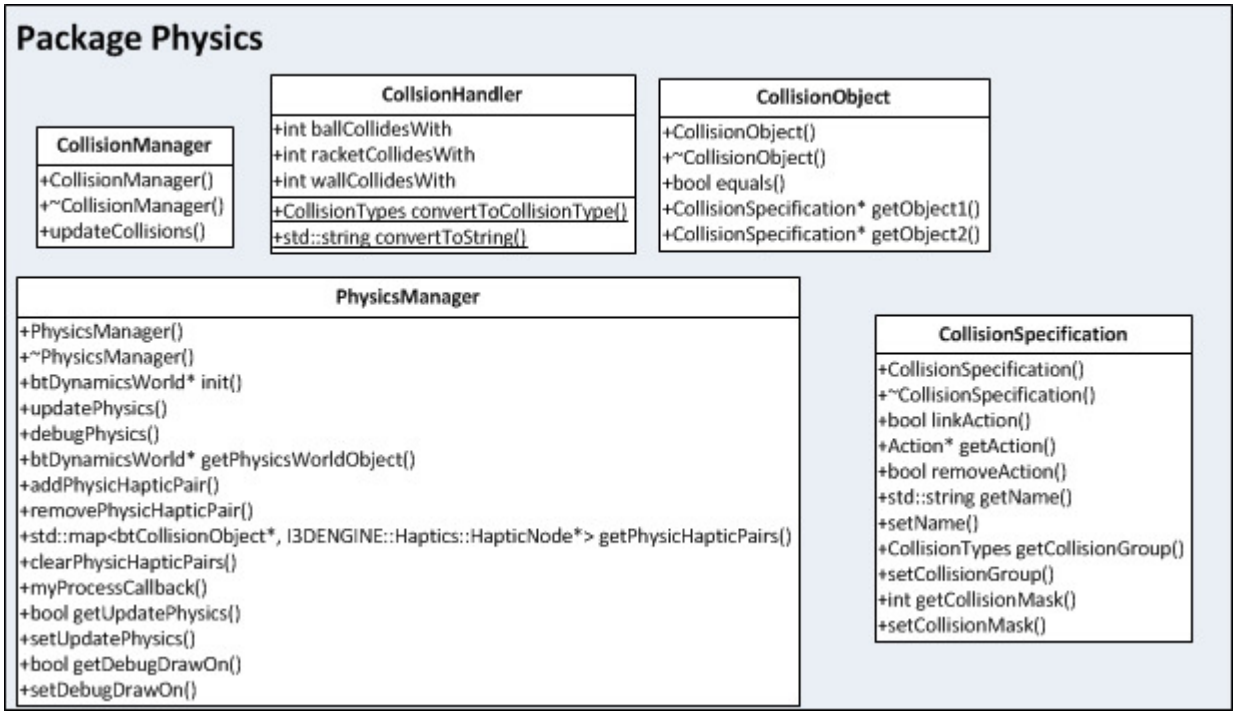


Abbildung 36: UML-Diagramm der Physik-Klassen

PhysicsManager:
Der PhysicsManager kapselt die Physik-Welt aus dem Framework Bullet. In dieser Klasse wird die Physik-Welt in Bullet initialisiert und verwaltet. Ebenfalls steuert der PhysicsManager das Updaten der Physik. Eine weitere Funktion betriff das Initialisieren und Verwalten des GLDebugDrawer. Mit dieser Funktion ist es möglich, die Objekte von Bullet im OSG zu zeichnen und visuell darzustellen. Der PhysicsManager untersucht auftretende Kollisionen auf ihre Gültigkeit und sendet sie gegebenenfalls an den CollisionManager.
Der PhysicsManger läuft nur, wenn der HapticManger zur Verfügung steht. Dieser wird nur initialisiert, wenn ein Haptic-Gerät angeschlossen und gefunden wird. Damit wird sichergestellt,

dass im CAVE Kollisionen nur auf einer Node (Haptic-Computer) ausgelöst werden und nicht auch noch durch die anderen Nodes.

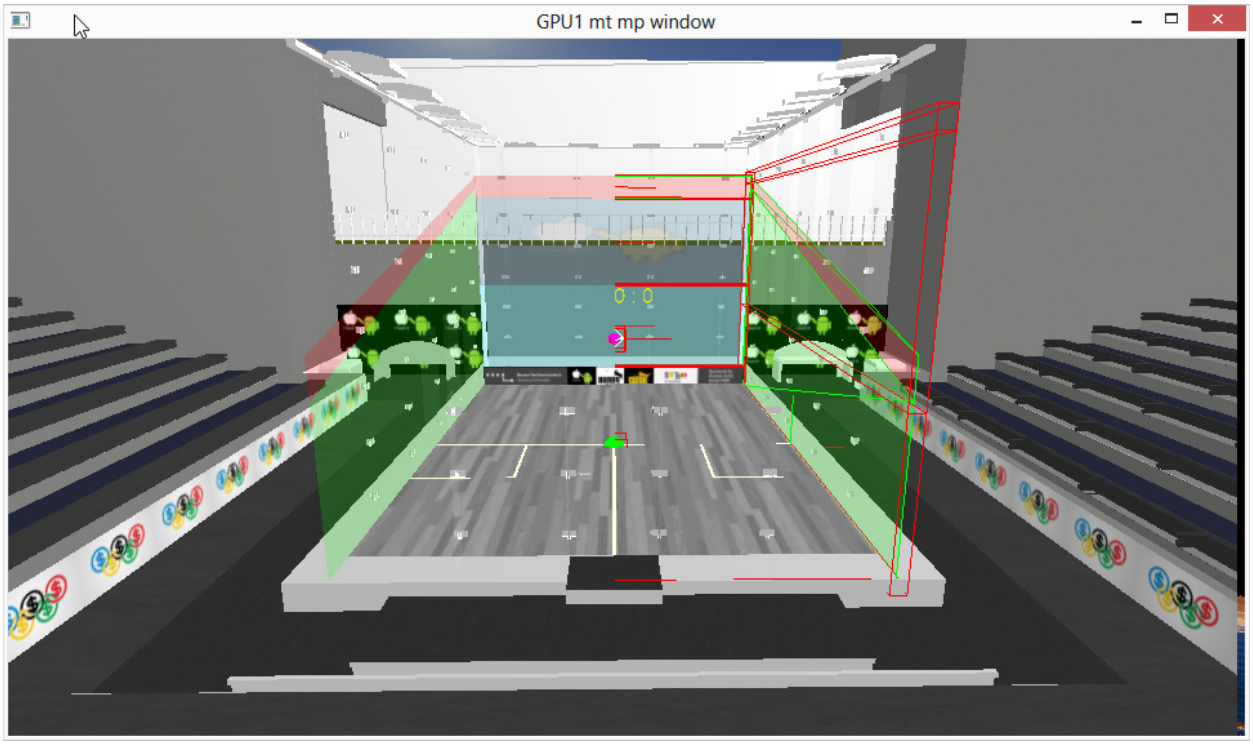


Abbildung 37: Die Physik-Welt von Bullet.

Links ohne diese anzuzeigen, rechts wird diese durch den DebugDrawer gezeichnet.

CollisionHandler:
Dies ist eine Hilfsklasse, welche einige Konstanten definiert, die in dem Umgang mit Kollisionen nützlich sind. Hier ist der CollisioneTypes definiert, welcher die Kollisionsgruppen darstellt. Diese Klasse macht stellt nur die Konstanten zur Verfügung.

CollisionSpecification:
Diese Klasse wird verwendet, um wichtige Informationen für die Kollisionserkennung abzuspeichern. Ein wichtiges Attribut dieser Klasse ist die _collisionGroup, welche die Gruppe der Kollision abspeichert. Der Ball zum Beispiel gehört der Gruppe "COL_BALL" an, alle Bereiche im Squash-Court die als Out gelten gehören zur Gruppe "COL_OUT". Anhand dieser Eigenschaft kann ein Objekt die Aktion anhand der Gruppe des anderen Objektes ausführen.

Alle möglichen Aktionen eines Objektes sind ebenfalls in dieser Klasse gespeichert und jeweils mit einer Kollisions-Gruppe verknüpft. Auf einem Objekt kann für jede Kollisions-Gruppe keine oder eine Aktion gespeichert werden. Die wird dann ausgeführt, sobald eine Kollision mit dieser Gruppe passiert.

CollisionObject:
Objekte der Klasse CollisionObject werden vom PhysicManager erstellt und als Liste von momentan auftretenden Kollisionen an den CollisionManager geschickt. Das CollisionObject ist eine kleine Hilfsklasse, welche von beiden Kollisions-Objekten (z.B. Ball und Seitenwand) die CollisionSpecification beinhaltet.

CollisionManager:
Der CollisionManager verwaltet in einer Liste alle momentan auftretenden Kollisionen und aktualisiert die Liste nach jedem Physik-Update. Für alle neu auftretenden Kollisionen wird in dieser Klasse ein CollisionEvent erstellt und verschickt.

4.7.9 Aktionen für die Logik

Unsere Grundidee für dieses Spiel ist, dass dieses mit haptischen Geräten (speziell dem Phantom Omni) gesteuert werden kann. So muss das Menu, sowie das Spiel selber, fähig sein, damit umzugehen. Jedoch sollte der Spieler noch die Möglichkeit haben, mit der Tastatur gewisse Aktionen auszuführen, so zum Beispiel durch das Menu navigieren. Diese Voraussetzung konnten wir mit dem EventSystem schaffen. Tastatureingaben werden, falls definiert, als entsprechender MenuEvent verschickt und können so für die Navigation durch das Menu verwendet werden. Mit Hilfe der Kollisions-Erkennung des PhysicsManager‘s und CollisionManager‘s werden ebenfalls die korrekten Events der Haptic verschickt und lassen so eine Steuerung des Menu’s zu.

Implementiert Feature (siehe Pflichtenheft):

- 6.5.13 Audio

4.7.9.1 Aktionen oder Events

Für unsere Architektur haben wir folgenden Grundsatz durchgezogen, wann wir welche Klasse einsetzten und wie wir diese definiert haben:

Aktionen

Eine Aktion wird dann verwendet, wenn am lokalen Computer etwas gemacht werden muss. So hat zum Beispiel jeder Menu-Eintrag im Menu eine MenuAction. Sobald der Befehl gegeben wird, dass der aktuelle Menu-Eintrag ausgeführt wird, wird auf dem Menu-Eintrag die Aktion ausgeführt. Die Aktion beinhaltet, dass ein neues Menu geladen oder in die Spielhalle zum Spiel gewechselt wird. Je nach Aktion auf dem Menu-Eintrag wird dann ein Event losgeschickt, damit alle anderen Computer im CAVE ebenfalls das gleiche machen können: entweder ein neues Menu oder das Spiel laden.

Events

Ein Event wird dann verwendet, sobald etwas über die Event-Queue an die anderen Computer verschickt wird. Ein neues Menu laden mit der entsprechenden Menu-Id ist ein solcher Event. Sobald ein neues Menu geladen werden soll, wird ein Event, mit der Id des neuen Menu’s, an alle Computer versendet. So kann dann jeder Computer das neue Menu laden.

In unserem System werden die Actions nur von den verschiedenen Kollisions-Events sowie von gewissen Tastaturevents (bezüglich Menuführung) aufgerufen. Andere Events haben mit den Actions nichts zu tun.

Schlussfolgerung

Es ist also durchaus möglich, dass eine Aktion einen Event generiert, um eine entsprechende Meldung an alle Computer zu machen. So kann gewährleistet werden, dass überall immer derselbe Stand des Spieles ist. Ebenfalls ist es möglich, dass ein Event eine Aktion auf einem OSG-Element ausführen kann. Wird zum Beispiel die Enter-Taste gedrückt im Menu, wird ein MenuEvent geschickt mit der Meldung dass das aktuelle Element ausgeführt werden soll. Der EventHandler führt dann die entsprechende Aktion auf diesem Element aus.

4.7.9.2 Aktionen auf den Objekten

Jedes virtuelle Objekt (z.B. Ball, Racket, Wand, etc.), welches für die Menu- oder Spiellogik von Bedeutung ist, hat ein CollisionSpecification als Attribut. In diesem können die verschiedenen Aktionen gespeichert werden. Zu jeder Aktion, eines solchen Attributes, wird zudem gespeichert, auf welches andere Kollisions-Objekt diese Aktion reagieren soll. Auf dem Boden ist eine Aktion CollisionAction hinterlegt, welche nur dann ausgeführt wird, wenn das andere Kollisionsobjekt der Ball ist. Ist das andere Objekt das Racket, wird keine Aktion ausgeführt.

Mit dem geladenen EventHandler stellen wir richtig, dass die entsprechenden Aktionen ausgeführt werden, wenn ein Kollisions-Event erkannt wird. Ebenfalls wird der Event von der Tastatur entsprechend versendet um korrekt erkannt zu werden.

Ein Beispiel:

Ein CollisionEvent beinhaltet zwei Strings. Beide Strings repräsentieren je ein Objekt im Scenegraph. Eine Kollision zwischen dem Ball und dem Boden generiert also einen CollisionEvent mit den beiden Strings "ball" und "floor". Wenn der Event beim EventHandler für das Spiel ankommt, werden beide Objekte im Scenegraph gesucht. Wenn beide Objekte gefunden wurden, werden die Aktionen gesucht, die mit dem anderen Objekt verknüpft sind. So wird auf dem Ball eine Aktion gesucht, welche mit dem Boden verknüpft ist und auf dem Boden eine Aktion, die mit dem Ball verknüpft ist. Wird eine Aktion gefunden, wird diese mit dem entsprechenden Befehl action.execute() ausgeführt. Mit diesem Aufbau spielt es keine Rolle, auf welchem Objekt die Aktion hinterlegt wurde. Wichtig ist nur, dass diese mit dem entsprechenden anderen Objekt verknüpft wurde.
Je nachdem welche Aktion auf den Objekten hinterlegt ist, wird wieder ein anderer Event verschickt. Es kann zum Beispiel sein, dass wieder zurück ins Menu gewechselt wird, sobald ein Spieler gewonnen hat.

4.7.9.3 Implementation

Die Logik der Actions ist auf dem Command-Pattern aufgebaut und für uns angepasst. Das typische UML-Diagramm für das Command-Pattern ist folgendes:

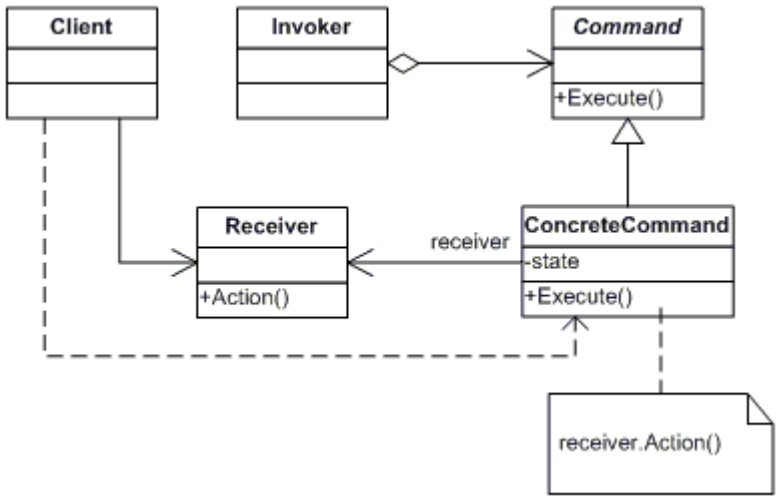


Abbildung 38: Klassisches Command-Pattern

In unserem System sieht das UML für die Aktionen, wie auf dem Diagramm unten aus. Bei diesem UML Diagramm werden nur die Events (CollisionEvent und MenuEvent) dargestellt, welche unter anderem Actions auf den OSG-Objekten aufrufen.

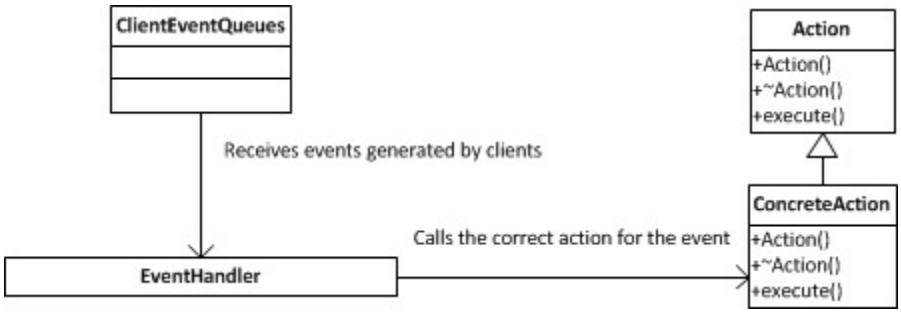


Abbildung 39: UML Diagramm des Action-System

Unsere Aktionen gesamthaft werden in folgendem UML Diagramm dargestellt und werden anschliessend beschrieben:

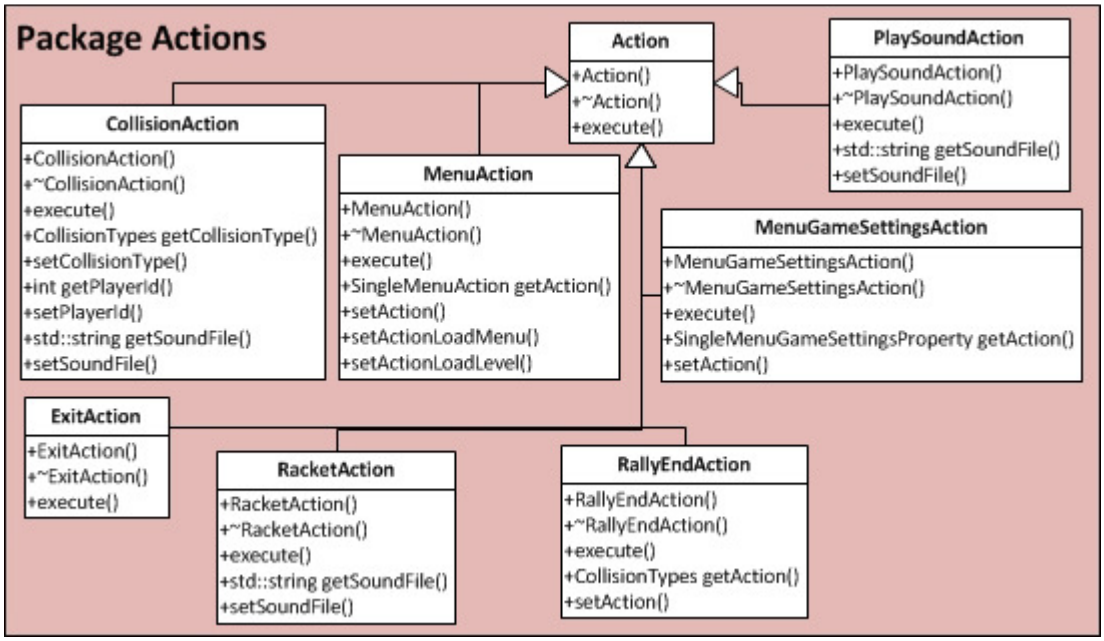


Abbildung 40: UML-Diagramm mit allen Actions.

CollisionAction:

Die CollisionAction wird im Spiel gebraucht. Diese definiert das Verhalten einer Kollision zwischen dem Ball und einem anderen Objekt. Die Kollision zwischen dem Racket und einem Objekt wird nicht mit einer CollisionAction abgedeckt. Die CollisionAction ermittelt den nächsten Status für die BallStateMachine. Aus diesem Grund muss die CollisionAction auf dem Objekt sein, welches den nächsten Status repräsentiert, so zum Beispiel auf dem Boden und nicht auf dem Ball. Mit diesem neuen Stand wird die BallStateMachine aktualisiert. Fällt diese in einen ungültigen Status wird entsprechend reagiert. Die neue Punktezahl für die Spieler wird ermittelt und mit einem Event wird der Ball an seinen Ursprungspunkt zurückgesetzt und die BallStateMachine wird zurückgesetzt.

Wenn zusätzlich der Action noch einen Sound hinzugefügt wurde, wird der entsprechende Sound dem Ball angehängt und abgespielt.

MenuAction:

Mit einer MenuAction kann definiert werden, was auf einem Menu-Punkt passieren soll wenn dieser bestätigt wird. Um durch ein Menu zu navigieren und das Spiel zu laden genügen die folgenden zwei Möglichkeiten:

- Ein neues Menu mit der angegebenen Id laden.
  Der MenuAction muss die entsprechende Id angegeben werden. Sobald diese Action ausgeführt wird, wird ein Event an alle Nodes geschickt, um ein neues Menu zu laden. Um zurück in das vorherige Menu zu wechseln, muss ebenfalls ein Menu-Punkt erstellt werden, welcher das vorherige Menu lädt. Dies wird im XML definiert und ist in einem anderen Kapitel definiert. Siehe 4.7.6.5 Besonderheiten der Menu‘s

- Ein Spiel mit der angegebenen Id laden
  Um das Spiel zu laden, wird ein Event an alle Nodes geschickt, dass das Level mit der angegebenen Id geladen werden soll. Die Id des Levels wird entweder in der XML bereits fix definiert oder falls dies nicht im XML definiert wurde, wird automatisch die aktuelle Id genommen, welche auf den GameSettings gespeichert ist.

ExitAction:

Die ExitAction beendet das Spiel und schliesst dieses ab.

PlaySoundAction:

Die PlaySoundAction spielt lediglich einen Sound ab, welcher für diese Action definiert wurde.

MenuGameSettingAction:

Mit der MenuGameSettingsAction können die Einstellungen der aktuellen GameSettings verändert werden. Dazu muss der Action übergeben werden, welche Einstellung geändert werden soll. Die verschiedenen Einstellungsmöglichkeiten sind in einem Enum gespeichert und können entsprechend gesetzt werden. Zudem muss der zu verändernde Wert gespeichert werden. Für den Wert gibt es zwei verschiedene Möglichkeiten:

- Der Wert stellt eine relative Veränderung dar. Diese kann negativ oder positiv sein. Somit wird der Wert dem aktuellen Wert hinzugezählt.

Der Wert stellt eine absolute Veränderung dar. Diese kann nur positiv sein, weil die Eigenschaften in der Klasse GameSettings nur positive Werte erlauben. Wird eine solche Aktion ausgeführt, wird der alte Wert einfach mit dem Neuen überschrieben.

4.7.10 Haptic

Mit der Haptic haben wir uns in der Arbeit relativ lange und intensiv beschäftigt.

4.7.10.1 Planung

Ein grosser Teil unserer Arbeit nimmt die Umsetzung der Haptic in Anspruch. Bei der Haptic legen wir Wert darauf, dass dieses sauber implementiert ist und der Input der Geräte korrekt an alle Clients weitergegeben wird. Damit wird gewährleistet, dass alle Clients den Scenegraph korrekt updaten können und der Spielstand auf allen Clients konsistent ist. Der Output der haptischen Geräte muss nur auf dem Computer gerechnet werden, an welchem das Gerät direkt angeschlossen ist. Mit Hilfe dieser Architektur ist es möglich, das Haptic-Gerät an einem beliebigen Computer in diesem Netzwerk anzuschliessen.

Generelles

Die Integration von der Haptic im I3D wurde bereits in einer anderen Bachelor-Arbeit umgesetzt. Auf dieser werden wir unsere Arbeit aufbauen um die Haptic in einem 3D-Spiel zu verwenden. Die Haptic ist bereits ein Bestandteil des Frameworks I3D. Änderungen am Verhalten der Haptic-Klassen werden wir nicht direkt im Framework I3D vornehmen, sondern, wenn möglich, Vererbung einsetzen. In unserem Fall werden wir alle Haptic-Klassen im Projekt-Ordner "Haptics" unterbringen.

Haptic Pointer: Mesh

Bisher ist im I3D die Haptic mit einer Kugel dargestellt. SquashI3D soll jedoch als Modell ein Racket einsetzen. Aus diesem Grund werden wir ein Mesh von einem Racket erstellen und als Pointer für die Haptic darstellen. Um dies zu erreichen, müssen wir einiges an der Vererbung der verschiedenen Klassen, wie z.B. im Generic3dofPointer, vornehmen. Das I3D-Framework Klasse Generic3dofPointer hat sehr viele statische Abhängigkeiten auf z.B. den HapticBulletSpherePointer. Diese Verhalten müssen angepasst werden.
Nachfolgend eine Übersicht der anzupassenden Eigenschaften:

- Statische Verknüpfung mit dem HapticBulletSpherePointer
- Voraussetzung, dass der Haptic direkt am OSG-Root-Knoten angehängt ist
- Voraussetzung, dass nur ein haptisches Gerät im Einsatz steht.

Für eine Beispielapplikation kann diese Art Pointer durchaus Sinn machen. Jedoch sind solche statische Werte für ein Framework eher ungeeignet. Für die Arbeit mit einem Mesh muss ein generischer Ansatz ausgearbeitet werden. Der SquashI3D Generic3dofPointer soll somit mit einer abstrakten HapticBulletPointer-Klasse arbeiten können, welcher dann die Implementation eines Meshes oder der speziellen Kugel repräsentieren kann.

Singleplayer

Unser erstes Ziel ist es, das Spiel mit einem Haptic-Gerät zu implementieren. Dies wollen wir mit einer einfachen Architektur erreichen, in der wir das Haptic-Gerät direkt am Master anschliessen. Dieser ist für die Synchronisierung zuständig. Nachfolgend ein Diagramm, welches kurz aufzeigt, welche Synchronisierungen zwischen Input und Output verwendet werden. Um das Diagramm einfach zu halten, werden nur 3 Clients anstatt der 8 Clients im CAVE dargestellt.

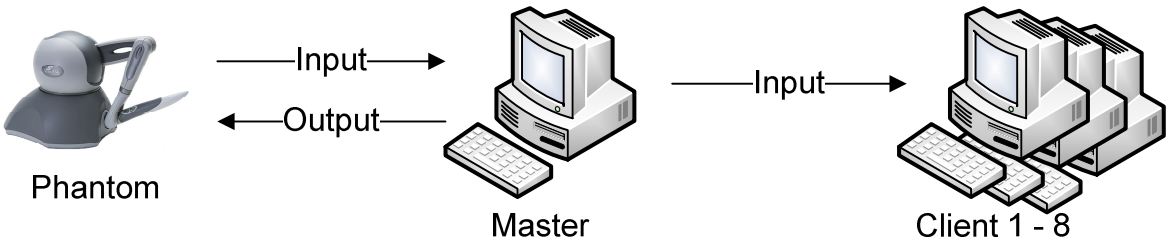


Abbildung 41: Input und Output der Haptic Einzelspieler

Multiplayer

Unser Multiplayer Spiel wird mit maximal zwei Spielern gespielt. Dazu muss natürlich jeder Spieler sein eigenes Haptic-Gerät besitzen um seinen Schläger zu steuern. Unsere Idee ist es, beide Haptic-Geräte am gleichen Computer anzuschliessen, um so den HapticRenderThread nur auf einem Computer laufen zu lassen und die Synchronisierung zu vereinfachen. Somit ist die Erweiterung zum Singleplayer Spiel nur eine minimale.

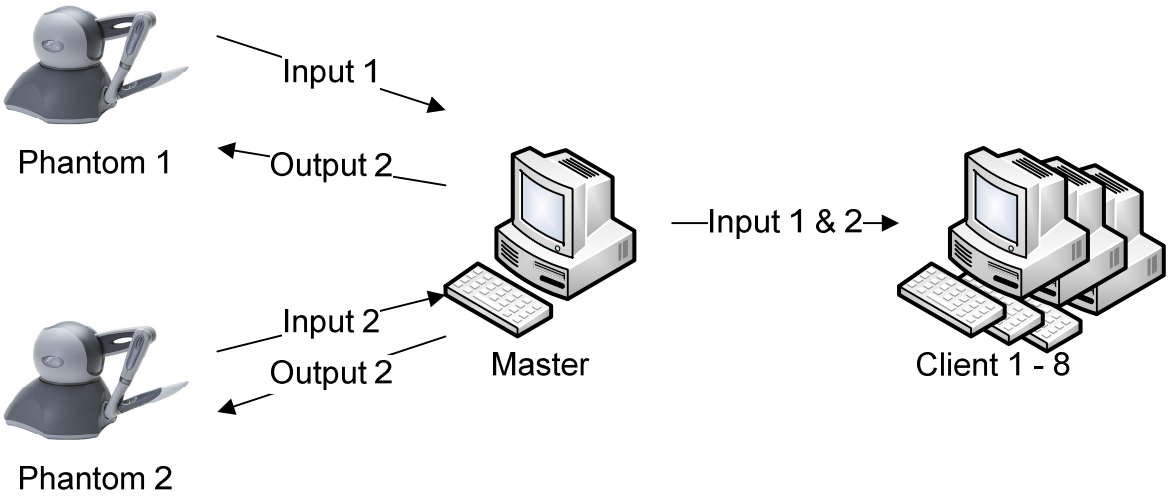


Abbildung 42: Input und Output der Haptic Mehrspieler

In einem Gespräch mit Herrn Künzler über die Umsetzung der Haptic mit mehreren Geräten sind wir noch zu einer zweiten Variante gekommen. In dieser wird pro Haptic-Gerät ein zusätzlicher Client im CAVE-System angeschlossen. Diese neuen Haptic-Computer haben je einen HapticRenderThread und haben die volle Kapazität, um für das angeschlossene Haptic-

Gerät den Input, Output inklusive physikalische Berechnungen durchzuführen. Die Synchronisation muss gewährleistet werden zwischen allen anderen Clients.

Für diese Variante gilt folgendes Diagramm:

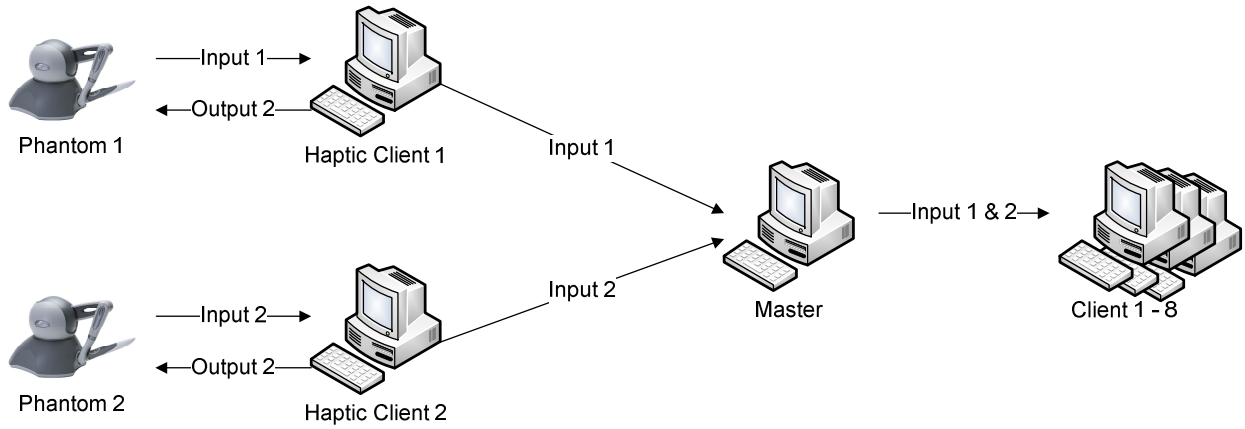


Abbildung 43: Input und Output der Haptic Mehrspieler über Netzwerk

Wir haben uns für die Variante 1 entschieden, damit wir uns nicht zusätzlich Gedanken zur Synchronisierung der verschiedenen Haptic Clients machen müssen. Ein weiterer Grund für die Variante 1 ist, dass wir keine Verzögerung mit dem Input und dem Feedback haben. Wird die Haptic auf einem weiteren Computer ausgelagert, so besteht die Möglichkeit, dass man mit dem Haptic-Server jeweils ein Frame hinter her liegt. Dieses Verhalten müsste jedoch genauer untersucht werden.

4.7.10.2 Umsetzung

Generelles

Die Haptic ist bereits im I3D integriert. In einer ersten Version haben wir diese übernommen. Jedoch stellten wir fest, dass viele Anforderungen von SquashI3D damit nicht abzudecken sind. Das Problem, der bisher integrierten Version der Haptic, ist, dass alles statisch auf das Modell "Sphere" programmiert wurde. Wird ein anderes Objekt geladen, funktioniert die I3D-Haptic nicht mehr korrekt. Was wir zu einem späteren Zeitpunkt ebenfalls feststellten, ist, dass die Algorithmen für die Kollisionserkennung ebenfalls nur für diese Kugeln geschrieben sind und nicht dynamisch genug erweiterbar sind. Vieles ist statisch für den Fall Kugel implementiert. Als Beispiel: Werden die Haptic-Pointer‘s nicht mehr direkt am OSG-Root-Knoten angehängt, findet der Algorithmus keine Kollisionen mehr.

Um deshalb mehr Einfluss auf die Haptic und deren Implementierung zu nehmen zu können, haben wir viele Klassen für die Haptic-Programmierung in unserem Projekt neu definiert.

Haptic Pointer: Mesh

Das Laden und verwalten des Meshes geschieht in der Klasse "HapticBulletPointerMesh" implementier. Die Klasse leitet von der Klasse "HapticBulletPointer" ab, welche mit dem Generic3dofPointer zusammen funktioniert. Somit kann das Mesh durch diese verwaltet werden. Aus dem geladenen Mesh im OSG wird ebenfalls noch ein Kollisions-Objekt für Bullet erstellt, um Kollisionen mit dem Mesh zu erkennen.

Singleplayer

Das Ansteuern eines Haptic-Gerätes hat gut und schnell funktioniert. Die Initialisierung des Gerätes im HapticManager erfolgte einwandfrei. Das Gerät wurde gut erkannt und sobald das Haptic-Gerät am Computer angeschlossen war, erkannte das SquashI3D dies und zeigte den definierten Haptic-Pointer an, welcher das Gerät im 3D-Raum repräsentierte. Wie bereits erwähnt, wird in unserem Fall ein Squash-Racket als Mesh geladen. Dieser übernimmt die Lage (Rotation) und Position (Translation) des Gerätes. Die Kollisionen werden entsprechend mit dem PhysicManager erfolgreich erkannt.
Das entsprechende Feedback der Kollisionen können wir durch verschiedene Änderungen des Pointers nicht mehr korrekt an das Gerät übermitteln (siehe 5.6 PotentialFieldForceAlgo und ProxyPointForceAlgo). Um zu verhindern, dass das Gerät Schaden annimmt, haben wir uns entschieden, das Feedback von den Kollisionen auszulassen.

Die Synchronsierung der verschiedenen Nodes über den Equalizer muss jeweils im CAVE getestet werden.

Mutliplayer

Wir haben entschieden, dass wir die Variante mit maximal einem Haptic-Computer pro System (Haptic-Geräte am Master) vorziehen. Dies ergibt folgende neue Anforderungen:

HapticManager

- Unterstützung von x-beliebigen Haptic-Geräten

SceneRenderThread

- Unterstützung von x-beliebigen Haptic-Geräten

Das Erkennen von mehr als einem Haptic-Geräte machte zusätzliche Schwierigkeiten, wie in 5.4 Haptic-Anbindung nachzulesen ist. Einen Performance-Verlust durch das Verwenden von zwei Geräten, konnte nicht festgestellt werden. Das Testen mit zwei Geräten funktionierte spürbar gleich schnell wie mit nur einem Gerät.

4.7.10.3 Implementation

Die Implementation ist grösstenteils der bestehenden Haptic-Integration des I3D nachempfunden. Wie bereits unter 4.7.10.2 Umsetzung erwähnt, sind die Klassenim Package "Haptics" zu finden. Die Vererbung musste überarbeitet werden, damit der zu verwendende Haptic-Pointer in Zukunft dynamisch weiterentwickelt oder ein neuer hinzugefügt werden kann.

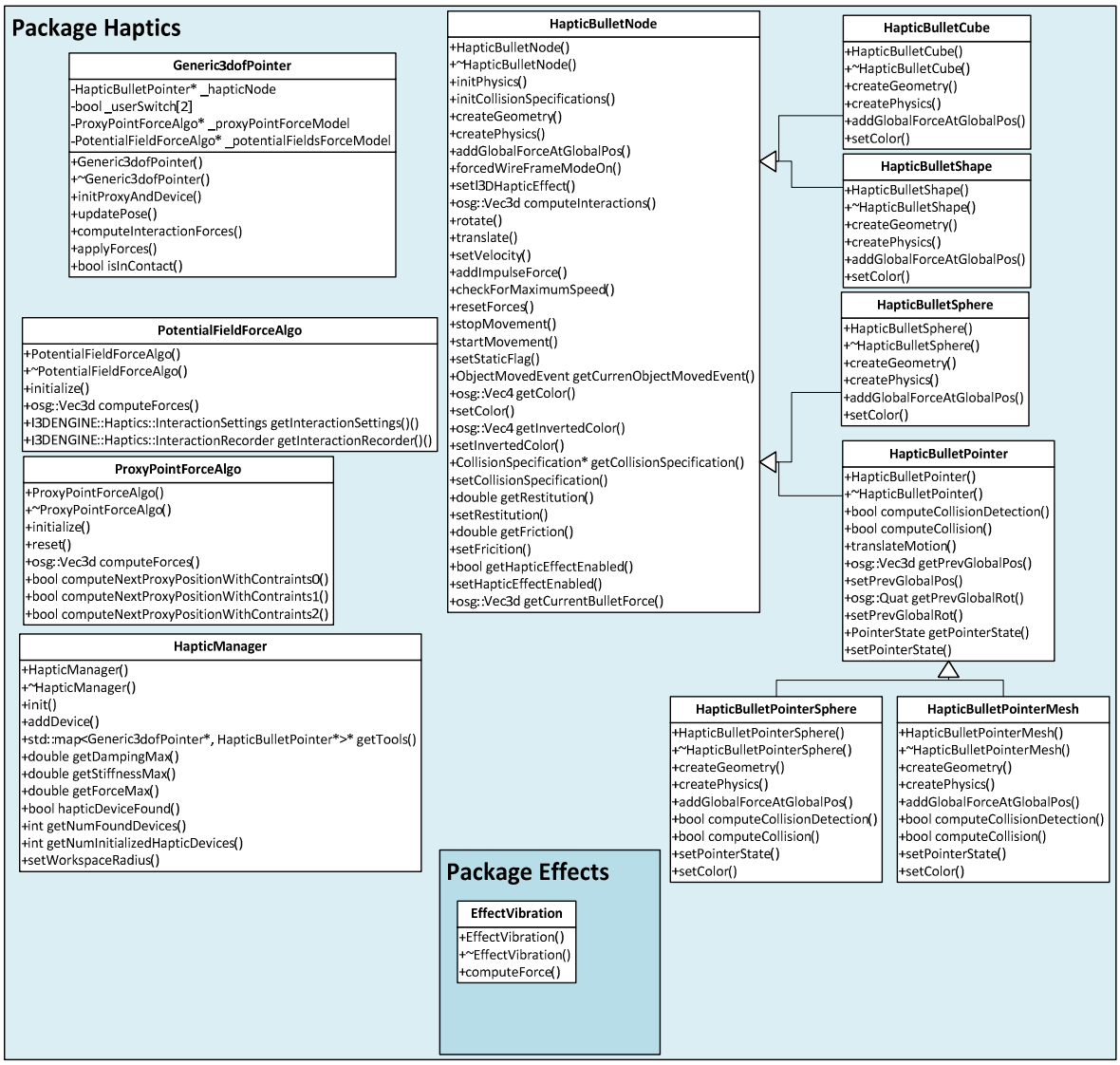


Abbildung 44: UML-Diagramm des Haptic-Packages.

HapticManager:

Der HapticManager verwaltet alle Referenzen und Eigenschaften der initialisierten Geräte. Zudem erkennt er neue nicht initialisierte Geräte und kann diese entsprechend nachladen. Dies geschieht durch das Framework Chai3D, welches hier genutzt wird.

Der HapticManger bleibt nur vorhanden, wenn beim Starten des SquashI3D ein lauffähiges Haptic-Gerät am Computer angeschlossen wurde. Ansonsten wird dieser wieder gelöscht.

HapticBulletNode:

Diese Klasse leitet von einem OSG-Knoten ab, und kann somit einfach im SceneGraph integriert werden. Zusätzlich zu den durch OSG gegebenen grafischen Funktionalitäten werden in dieser Klasse die Bullet-Eigenschaften initialisiert und verwaltet. Das CollisionShape für Bullet wird erstellt, je nach Implementation in der abgeleiteten Klasse, und dem PhysicsManager übergeben, um die nötigen Referenzen und Eigenschaften für die Kollisionserkennung bereitzustellen. Weitere Informationen, welche die Kollisionserkennungen betreffen, werden ebenfalls auf dieser Klasse verwaltet. Eine entsprechende Initialisierungsmethode ist mit "initCollisionSpecification" gegeben.

HapticBulletShape / HapticBulletCube / HapticBulletSphere:

All diese Klassen sind konkrete Implementierungen der Klasse "HapticBulletNode" und besitzen die benötigten Implementationen, um jeweils einen Cube, Sphere oder ein sonstiges Shape von OSG im SceneGraph darzustellen. Zudem werden die entsprechenden CollisionShapes für Bullet erstellt und verwaltet. Die weiteren von uns benötigten Informationen für die Kollisionen werden ebenfalls auf der Basis-Klasse initialisiert (wird durch den Level-/MenuCreator vorgenommen => 4.7.6. Level und Menu)

HapticBulletPointer:

Mit dieser abstrakten Klasse ist es möglich, den OSG-Knoten (inklusive Bullet) mit dem Phantom durch den Raum zu steuern. Die Rotation und Translation wird entsprechend übernommen. Den Rest kann durch die Basis-Klasse "HapticBulletNode" verwaltet werden.

HapticBulletPointerMesh / HapticBulletPointerSphere:

Diese Klassen sind konkrete Implementationen des HapticBulletPointer. Mit dem HapticBulletPointerMesh wird ermöglicht, ein frei wählbares Mesh in den Pointer zu laden und diesen mit dem Phantom im Raum zu navigieren. Die CollisionShape wird durch die osgbCollision-Library durch das OSG-Modell erstellt, um das Mesh in Bullet korrekt zu laden. Hier gilt die Besonderheit, dass dem HapticBulletPointerMesh zwei verschiedene Modelle übergeben werden können. Das erste Model ist für die grafische Darstellung und das zweite für die Generierung des Bullet-CollisionShapes. So kann für die Physik und deren Berechnungen ein vereinfachtes Modell geladen werden, was der Rechengeschwindigkeit zugutekommt. Mit dem HapticBulletPointerSphere ist es möglich, eine einfache Sphere im Raum mit dem Phantom zu steuern. Diese hat nicht die genau gleichen Funktionalitäten, wie der bereits existierende HapticBulletNodeSpherePointer vom I3D. Diese haben wir nicht vollständig übernommen, weil die Sphere nicht Priorität war für unsere Arbeit, sondern das Mesh um ein Racket darstellen zu können.

Generic3dofPointer:

Die Klasse wird verwendet, um einen HapticBulletPointer mit einem Haptic-Gerät zu verknüpfen und diesen mit der jeweiligen Position zu aktualisieren und entsprechend Feedback zu berechnen und zurück an das Gerät zu geben. Initialisiert wird das Gerät durch den HapticManager, den HapticBulletPointer durch den SceneManager und in dieser Klasse wird nur ein Gerät auf einen HapticBulletPointer gemappt.
Diese Klasse haben wir vom I3D übernommen und entsprechend angepasst. Hier wurde viel statisch integriert, was es beinahe unmöglich gemacht hat, einen anderen Pointer zu verwenden. Deshalb musste einiges angepasst werden. Zudem wird hier die Berechnung der

Kollisionen durch verschiedene Algorithmen gemacht (die wahrscheinlich vom Chai3D übernommen wurden). Dies ist der PotentialFieldForceAlgo und ProxyPointForceAlgo, welche für die Kollisionserkennung der Pointer, sowie der Berechnung des Feedbacks zuständig sind. Das Problem dieser Algorithmen ist einerseits, dass diese ebenfalls viele statische Verknüpfungen beinhalten und anderseits für eine Sphere optimiert sind. Somit können diese nicht mit einem Mesh verwendet werden (siehe 5.6 PotentialFieldForceAlgo und ProxyPointForceAlgo). Die Kollisionen werden durch Bullet berechnet.

PotentialFieldForceAlgo:

Ein Algorithmus, um die Kollision einer Sphere mit einem anderen Objekt zu erkennen, noch bevor die Kollision wirklich stattfindet. Dazu wird ein Schwellenwert für die Distanz benutzt. Durch diese Erkennung ist es möglich verschiedene Effekte zu erzeugen, welche nicht zwingend eine direkte Kollision benötigen. So ist der magnetische Effekt ein solcher, um bereits im Vorfeld die anzuziehende Kraft zu bestimmen. Leider ist dieser Algorithmus auf eine Kugel ausgelegt.

ProxyPointForceAlgo:

Dieser Algorithmus bestimmt effektive Kollisionen einer Sphere mit einem anderen Objekt im Raum und errechnet der Kraftvektor, welcher an das Gerät zurückgegeben werden muss. Leider ist auch dieser Algorithmus auf eine Kugel ausgelegt.

EffectVibration:

Mit dieser Klasse wollten wir prüfen, ob wir ein einfaches Feedback an das Haptic-Gerät zurückgeben können, sobald das Racket mit dem Ball kollidiert. Dies ist uns jedoch nicht gelungen. Ebenfalls bei den Versuchen im HelloI3D konnten wir keinen diesen Effekten funktionierend nachstellen. Diese wurden bei der Kollisionserkennungen praktisch ignoriert. Nach einigem Aufwand haben wir diese Priorität herunter gesetzt, weil es selbst in der Beispielapplikation des HelloI3D nicht funktionierte und es somit nicht an unserer Implementation lag.

5 Probleme und Lösungsentscheide

5.1 FrameData

Problembeschrieb:

Im SquashI3D unterscheiden wir von Informationen, welche pro Computer gebraucht werden und Informationen, welche pro View/ Pipe gebraucht werden. Pro Pipe bedeutet, dass der Computer mehrere Output Kanäle hat. Also beispielsweise zwei Bildschirme oder Beamer.
So werden Informationen, welche die Kamera (virtuelle Kamera, also Sicht wie es der Spieler sieht) verändert direkt auf der Pipe verarbeitet. Jedoch haben wir ebenfalls Informationen, welche nur pro Node (Computer) gebraucht werden. Hierbei geht es um die Spielstatus Informationen: Beispielsweise Punktestand, Status des Balls, Audio-Lautstärke, etc;
insgesamt alle SquashI3D CustomEvents, welche über das FrameData versendet werden. Werden diese direkt in der Pipe verarbeitet, so werden diese doppelt ausgeführt. Dies ist unnötig und sollte nicht gemacht werden.

Lösung:

Die SquashI3D CustomEvents werden auf dem FrameData in einer Liste von UserEvents (siehe 4.7.5 Event-System) verwaltet. Verarbeitet werden diese UserEvents auf dem SceneSetup.
Das FrameData wird mit einer Laufnummer FrameData::frameCounter versehen. Das FrameData hat zwar auch Identifikatoren, wie ID oder InstanceID jedoch bezeichnen diese Identifikatoren nicht die Laufnummer, sondern sind zur ganzen Laufzeit gleich.
Das SceneSetup prüft bei der Entgegennahme einer UserEvent-Liste (CustomEvents) die aktuelle Laufnummer. Ist diese nicht grösser als die zuletzt verarbeitete, so werden die Events nicht verarbeitet. Dies da sie bereits zuvor von einer anderen Pipe gesendet wurden.
Dieses Problem war unerwartet, da dies in den CAVE-Demo-Applikationen bisher nicht beachtet wurde.

5.2 EventHandler-Wechsel

Problembeschrieb:

EventHandler befindet sich auf dem SceneSetup. Er ist verantwortlich für die Abarbeitung der verschiedenen Events. Die Interpretation, wie ein Event-Type verarbeitet werden soll, ist abhängig vom Kontext. Grundsätzlich kennen wir im SquashI3D zwei Kontexte:

- Menu
- Game

Für jeden dieser Kontexte gibt es einen EventHandler. Das Problem ist nun, dass ein EventHandler, während er die Events verarbeitet, sich selber nicht austauschen kann. Dies muss anders geschehen.

Lösung:

Der EventHandler befindet sich auf dem SceneSetup. Dieses nimmt die Liste der CustomEvents entgegen. Jeder Event wird einzeln dem EventHandler übergeben. Das SceneSetup hat die Möglichkeit zu prüfen, ob einer der Events ein EventHandlerChangedEvent ist. Das SceneSetup verarbeitet diesen Event und tauscht den Handler, falls notwendig, aus.
Alle anderen Events werden durch den EventHandler verarbeitet.
An diesem Punkt haben wir während des Software-Designs einen Fehler gemacht. Mit dieser Lösung haben wir jedoch einen passablen Weg gefunden, welcher auch von der Architektur her vertretbar ist.

5.3 Probleme mit DLL PhantomIoLib42.dll

Problembeschrieb:

Die PhantomIoLib42.dll gehört zu den Treiber-Dateien des SensAble Phantom Omni. Bei der Installation des Treibers wird diese DLL als 32-Bit und 64-Bit abgelegt.
Zum I3D-Framework-Packet gehört ein sogenanntes Install-Projekt. Dieses Projekt sucht sich alle abhängigen DLL's und speichert sie in den Ausgabe-Ordner. Damit hat man eine lauffähige Applikation.
Bei unseren ersten Tests haben wir diesen Prozess von Hand gemacht, weshalb wir alle Prototypen erfolgreich testen konnten.
Ab Dezember fiel uns auf, dass die Haptic im CAVE nicht mehr funktionierte. Wurde es jedoch lokal aus der Entwicklungsumgebung gestartet, so funktioniert die Haptic.

Lösung:

Der Grund, wieso die Applikation aus der Entwicklungsumgebung gestartet werden konnte, liegt im Auflösen der Assembly's. Wird eine DLL angesprochen, welche noch nicht in den Speicher geladen wurde, sucht Windows an folgenden Orten nach der Datei:

- Lokales Verzeichnis
- C:\Windows\System32 (für 32-Bit Systeme)
- C:\Windows\SysWOW64 (für 64-Bit Systeme)
- PATH Umgebungsvariable

Diese Verzeichnisse sind in der Umgebungsvariable PATH enthalten.

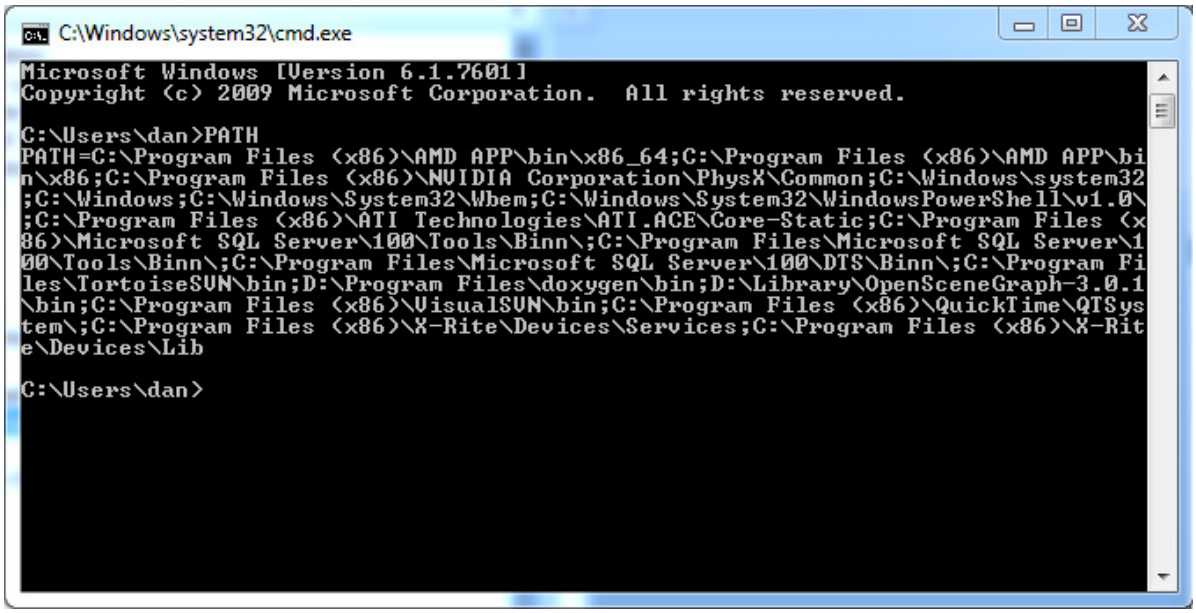


Abbildung 45: Umgebungsvariable PATH

Die Entwicklungsumgebung ergänzt diese PATH-Variable um folgende Verzeichnisse.

- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-Equalizer/debug/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DBASICS/_lib-xerces-c/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-osgBullet/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-OpenSceneGraphExts/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-osgWorks/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-glut/bin;
- **D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-chai3DExts;**

- D:/Projects/FH-Bern/SquashI3D/LIB-I3DVRDEVICES/_lib-osgVRPN/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DVRDEVICES/_lib-VRPN/debug/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DVRDEVICES/_lib-Intersense;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-SOFA/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-CUDA/bin;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DMULTIMEDIA/_lib-FMOD;
- D:/Projects/FH-Bern/SquashI3D/LIB-I3DENGINE/_lib-OpenSceneGraph/bin;)

Im Ordner **„/_lib-chai3DExts“** befindet sich die Datei „PhantomIoLib42.dll“. Diese war eine alte 32-Bit Version des Treibers. Bei der Installation des neusten Treibers auf unseren Entwicklungssystemen wurde eine neue Version des Treibers in „C:\Windows\System32“, rsp. „C:\Windows\SysWOW64“ kopiert. Dies bedeutet, solange im Root-Verzeichnis des Ausführungortes die Datei „PhantomIoLib42.dll“ nicht gefunden wurde, nahm Windows die Version aus „C:\Windows\SysWOW64“. Die SquashI3D-Applikation lief mit Haptic.
Erstellten wir die Applikation mit Hilfe des Install-Projektes, welches alle benötigten Dateien zusammenkopiert, so wurde die Datei „PhantomIoLib42.dll“ auf dem „**/_lib-chai3DExts**“ kopiert. Die Folge: Beim Auflösen der DLL wurde die Datei genommen, welche sich im Root-Verzeichnis befand. Dies war jedoch eine alte 32-Bit Version, was zu einem Absturz führte.

SquashI3D muss unter 64-Bit laufen und ist nicht 32-Bit kompatible.
Wir haben nun die „PhantomIoLib42.dll“-Datei im Verzeichnis „**/_lib-chai3DExts**“ durch die 64-Bit Version der neusten Treiber-Generation ersetzt.

Da praktische jede externe DLL als Fehlerquelle in Frage kam, kostete uns die Lösungsfindung rund 40h.

5.4 Haptic-Anbindung

Problembeschrieb:

Als wir zwei Geräte des Typs "PHANTOM Omni" an einem PC angeschlossen haben, funktionierte das Testprogramm von SensAble einwandfrei und beide Geräte konnten einzeln angesteuert und benutzt werden. In der Konfigurationsapplikation von SensAble für die Haptic-Geräte konnten beide Geräte hinzugefügt werden. Das erste Gerät ist automatisch mit "Default PHANToM" benennt. Das zweite Gerät haben wir mit "2nd Phantom" erstellt. Im I3D und in den Chai3D Beispielsapplikationen hatte es jedoch immer nur das Gerät gefunden, welches als Standard definiert wurde. Der Standard wird von SensAble automatisch durch alphabetische Sortierung der Gerätebenennung im Konfigurationsprogramm festgellegt. Um die Mehrspieler-Anforderung umsetzten zu können, mussten wir herausfinden, wieso nur ein Gerät erkannt wird.

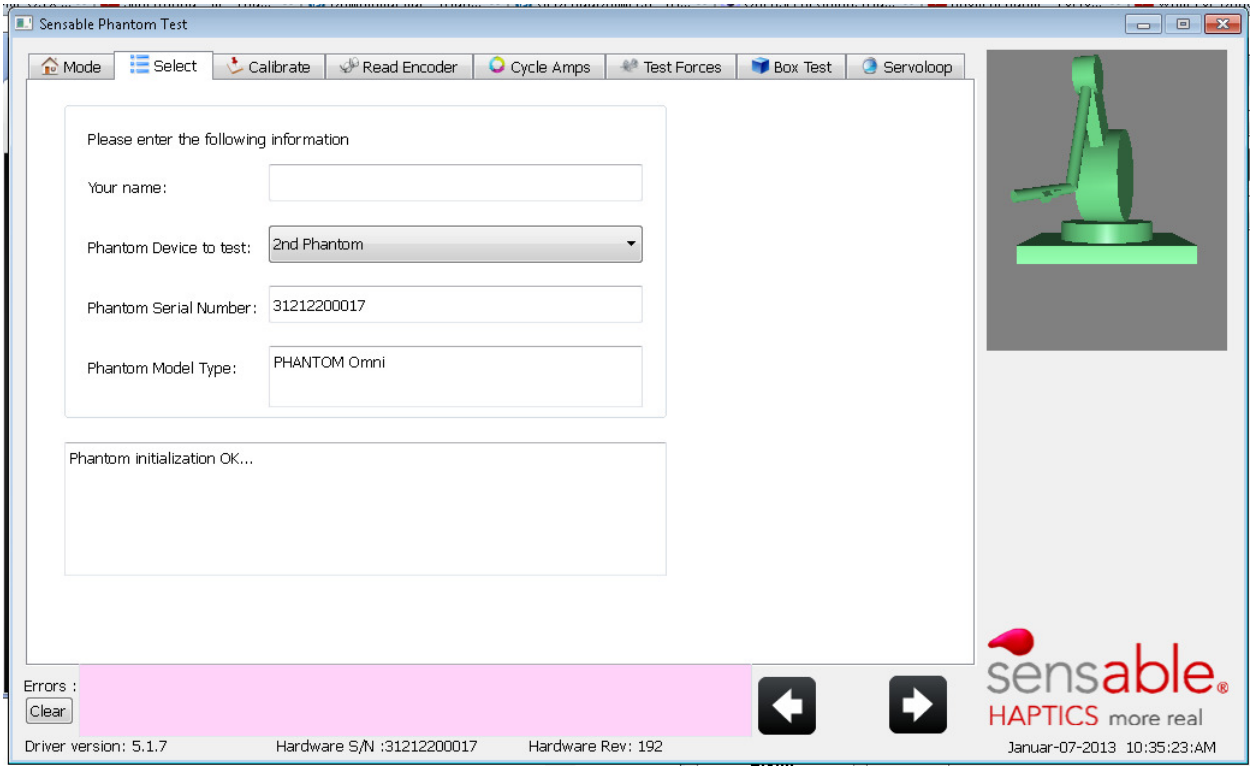


Abbildung 46: Beide Geräte können im Test-Programm angesteuert werden.

Lösungsfindung:

Im I3D wird die Implementierung zum Ansteuern mehrerer Geräte vom Chai3D übernommen. Nach Chai3D dürfen maximal 8 Haptische Geräte angeschlossen werden. Werden mehr als 8 Geräte angeschlossen, werden diese nicht erkannt und initialisiert. Für die Initialisierung ist die Klasse "cGenericHapticDevice" vom Chai3D zuständig. Für die in Chai3D definierten und erkannten Gerätetypen werden für die einzelnen alle angeschlossenen Geräte abgefragt und initialisiert. Das Überprüfen der Geräte wird in der entsprechenden DLL erledigt, welche pro Gerätetyp geladen wird. In unserem Fall für das Phantom Omni ist wird die DLL "hdPhantom.dll" geladen. Diese muss in einem entsprechenden Verzeichnis sein, welches von der Applikation erkannt wird um diese zu laden. Nur wenn die benannte DLL geladen werden kann, wird auch auf diesen Gerätetyp abgefragt. Dazu wird die Methode "getNumDevices" der

Klasse "cPhantomDevice" aufgerufen. Wenn kein solches Gerät angeschlossen ist, gibt die Methode die korrekte Zahl 0 zurück. Sobald ein Phantom am Computer angeschlossen ist, wird die korrekte Anzahl 1 gefunden. Doch bereits ab zwei verschiedenen Phantoms gibt die Methode immer noch den Wert 1 zurück.

Erste Schlussfolgerung:

Bereits hier konnten wir definieren, dass das Problem nicht direkt im Chai3D-Framework liegen kann. Die Programmierung im Chai3D greift auf die Funktionalität der hdPhantom.dll zurück. Der Fehler muss in dieser oder einer von ihr verwendeten Datei liegen, weil bei zwei und mehr Geräten hier nur ein Gerät gefunden wird.

Erste Versuche:

Die Initialisierung eines Phantom-Gerätes wird in Chai3D mit dem Befehl "new cPhantomDevice(index)" gestartet, welche nur den Index des Gerätes verlangt. Der Index ist eine ganze Zahl von 0 bis 8 (maximal definiert durch eine Konstante) und wird für jedes einzelne Gerät gegeben durch die fixe Sortierung von SensAble. Die Sortierung erfolgt anhand der Benennung der Geräte. Alphabetisch sortiert wird das erste Gerät mit dem Index 0 versehen. Für ein neues Gerät kann im SensAble Konfigurations-Tool einen neuen Namen vergeben werden. Später kann dieser in dem Programm nicht mehr umbenannt werden. Um ein bereits definiertes Gerät neu zu benennen, muss die entsprechende Config-Datei umbenannt werden. Diese Konfigurationsdateien befinden sich bei unserer Installation unter dem Ordner "C:\Users\Public\Documents\SensAble".
Der erste Versuch war zwei Geräte fix definiert zu laden und initialisieren. Somit haben wir ein neues Gerät mit "cPhantomDevice(1)" geladen. Dies hat leider nicht funktioniert und endete in einem Fehler, weil Chai3D das Gerät mit dem Index 1 nicht finden kann.

Weitere Schlussfolgerung:

Durch dieses Verhalten in Chai3D, was somit auch im I3D zutrifft, sind wir zum Schluss gekommen, dass es eindeutig ein Treiber-Problem in der Datei "hdPhandom.dll" ist, welche mit dem Chai3D mitgeliefert wird.

Weitere Lösungsfindung:

Weil in dem Testhaben wir noch mit Beispielen von SensAble versucht, zwei Geräte gleichzeitig anzusteuern. Damit die Haptic mit dem Computer sauber funktioniert, muss noch das OpenHaptic-Toolkit[2] von SensAble installiert sein. Dieses beinhaltet weitere Treiber und Beispiele, welche für die Funktionsweise relevant sind. Auf unserem Computer haben wir das unter dem vorgeschlagenen Verzeichnispfad installiert. Somit sind die Beispiele unter "C:\Program Files\SensAble\3DTouch\examples\bin\" zu finden. Viele Beispiele sind nur auf ein Haptic-Gerät ausgelegt. Jedoch gib es auch wenige Programme, welche zwei Haptic-Geräte verwalten, so zum Beispiel das Programm " HelloSphereDual.exe" oder "CoulombForceDual.exe". Beim ersten Ausführen erhielten wir nur eine Fehlermeldung:

[2] http://www.sensable.com/products-openhaptics-toolkit.htm

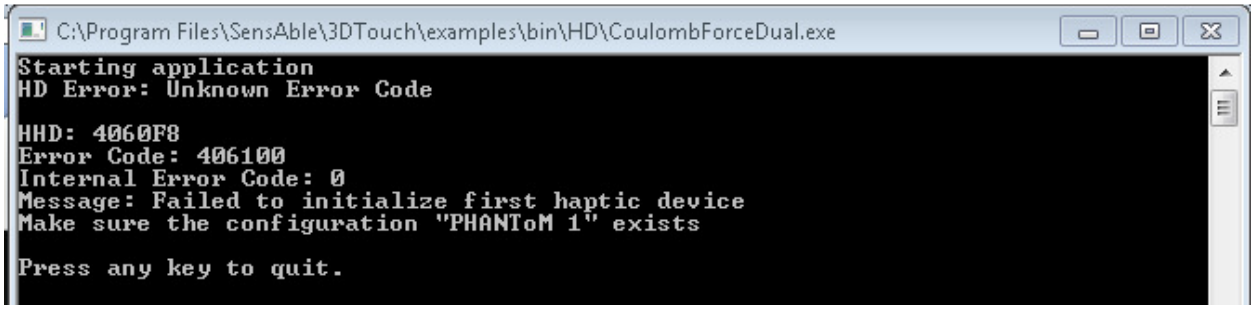


Abbildung 47: Erste Fehlermeldung für zwei Geräte

Erst als wir im Code von diesem Beispiel geschaut haben, waren wir sicher, dass die Fehlermeldung aufgerufen wird weil das Phantom-Gerät nicht gefunden wird. Aufgrund dessen haben wir festgestellt, dass im Beispiel versucht wird, das Gerät mit dem Namen "PHANToM 1" zu initialisieren, nicht mit einem Index. Im Konfigurations-Tool von SensAble ist es leider nicht möglich, bereits bestehende Konfigurationen umzubenennen. Dies mussten wir durch Umbenennen der Konfigurationsdateien im Ordner von SensAble machen. Die beiden Dateien müssen "PHANToM 1.config" und "PHANToM 2.config" heissen.

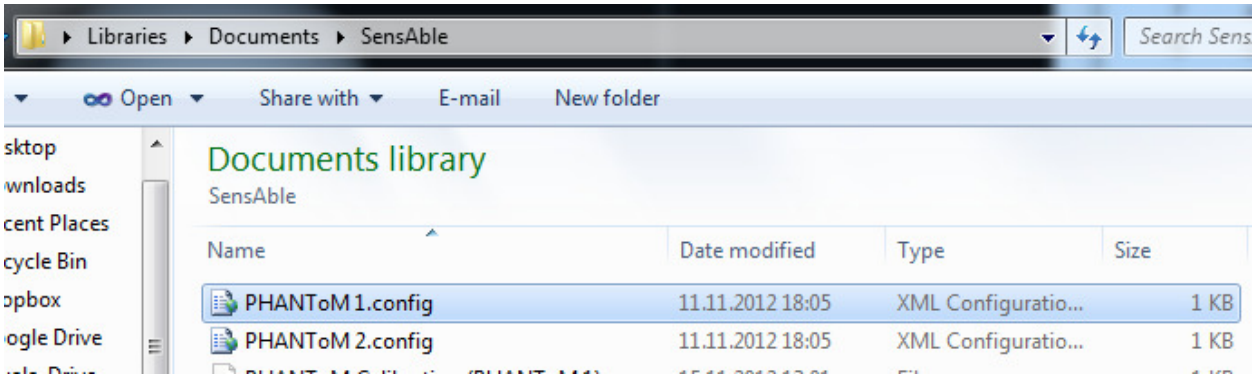


Abbildung 48: Richtige Bennenung der Config-Dateien für die Phantoms

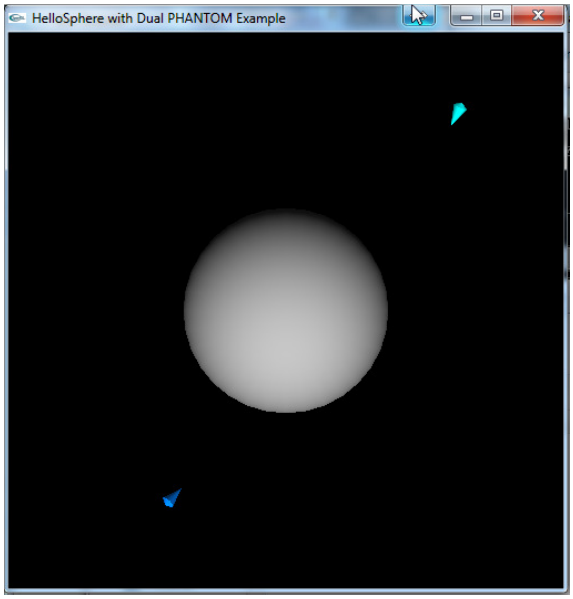


Abbildung 49: Beispiel SensAble-Demo mit zwei Phantoms

Weitere Lösungsfindung:

Somit ist unsere Vermutung bestätigt, dass es Probleme mit dem Framework Chai3D und der mitgelieferten DLL "hdPhantom.dll" gibt. Es wurde zwar angedacht, dass mehrere Geräte damit angesteuert und verwaltet werden können. Jedoch scheint es an der DLL "hdPhantom.dll" zu scheitern, welche scheinbar nur für ein Gerät ausgelegt ist. Zusätzlich haben wir verschiedene Konfigurationen der Haptic-Geräte versucht. Es spielt ebenfalls keine Rolle, ob ein Daisy-Chain gemacht wird mit FireWire oder ob beide Geräte direkt am Computer angeschlossen werden. Nur mit dem Beispiel von SensAble selber können beide Geräte gleichzeitig angesteuert werden.

Problemlösung 1: Haptic Clients

Eine Möglichkeit das Problem zu umgehen ist, wie bereits im Kapitel "4.7.10 Haptic" beschrieben, pro Haptic-Gerät einen eigenen Client aufzusetzen und mit dem Master-Computer zu synchronisieren. So müssten wir zusätzlich müssten wir so weitere Computer aufsetzten und entsprechend mit dem Master-Server synchronisieren lassen. Durch diese Synchronisation besteht die Möglichkeit, dass zwischen beiden Computern einige Frames Verzug sein kann. Für die Haptic wäre dies ein schlechter Effekt, weil diese schneller Wahrgenommen wird, als das Gesehene.

Problemlösung 2: Direktes ansteuern der SensAble-Treiber

Chai3D hat bereits angedacht, dass weitere Geräte selber in das Framework integriert werden können. Dazu bietet sich die Klasse " CMyCustomDevice" an. Diese muss die entsprechenden Methoden überschreiben und kann dementsprechend wie alle anderen Haptic-Geräte angesteuert und verwaltet werden. Dies hat den Vorteil, dass wir direkt das SensAble Framework ansteuern können, ohne dazu von der Datei "hdPhantom.dll" abhängig zu sein. Der Nachteil ist allerdings, dass wir ein weiteres Framework einbinden und erlernen müssen, um dies erreichen zu können.

Entscheidung:

Um uns zu entscheiden, wollten wir uns zuerst ein Bild machen, welche Lösung welchen Aufwand bedeutet. Um dies real abzuschätzen, haben wir kurz das Framework von SensAble studiert. Zudem wollten wir sehen, ob wir den Aufbau der Datei "hdPhantom.dll" herausfinden können, um abzuschätzen, was wir selber implementieren müssten. Somit haben wir im Ordner das Projekt für die "hdPhantom.dll" unter "EXT-CHAI3D\Source\external\hdPhantom\msvc9" gefunden und genauer studiert. In der Initialisierungs-Methode haben wir herausgefunden, wie nach den Phantom-Geräten gesucht wird. Schnell haben wir gesehen, wieso das Chai3D nur ein Gerät erkennt.

- Zuerst wird das Standard-Gerät initialisiert. Dazu wird der Wert NULL übergeben. Wird ein Gerät gefunden, wird die Anzahl Geräte um eines hochgezählt.
- Nach einem Kommentar " search for a possible second device" wird dann nach einem weiteren Gerät gesucht. Dazu wird der Wert "Phantom2" übergeben. Wird ein Gerät gefunden, wird die Anzahl Geräte um eines hochgezählt.
- Nach mehr Geräten wird nicht gesucht und der entsprechende Wert der Anzahl gefundener Geräte wird zurückgegeben.

Lösung:

Die Lösung um zwei Phantom-Geräte anzuschliessen ist also, das zweite Gerät "Phantom2" zu benennen. Zusätzlich muss das andere Gerät alphabetisch vor "Phantom2" gefunden werden können (Phantom1 ist erlaubt, Phantom3 nicht). Jedoch können somit nicht mehr als 2 Phantom-Geräte angeschlossen werden, weil nur für 2 Geräte geprüft wird. Sollen mehr als 2 Geräte initialisiert werden können, so muss diese "hdPhantom.dll" umgeschrieben werden, damit diese dynamischer nach den Geräten sucht.

5.5 C++ Header-Include-Strategie

Problembeschrieb

Da wir relative neu C++ entwickeln, haben wir uns anfangs nicht um eine Strategie, wie mit Header-Dateien umzugehen ist, gekümmert. Wir haben alle Includes direkt in Header-Files gemacht. Da wir auf verschiedene Frameworks zugreifen (I3D vereinigt Bullet, Chai3D, boost, etc.) haben wir mit der Zeit immer mehr Linker-Probleme erhalten.
Ein Linker-Problem kann entstehen, wenn sich verschiedene Dateien gegenseitig referenzieren. Sehr vereinfacht dargestellt, sieht das in etwa so aus:

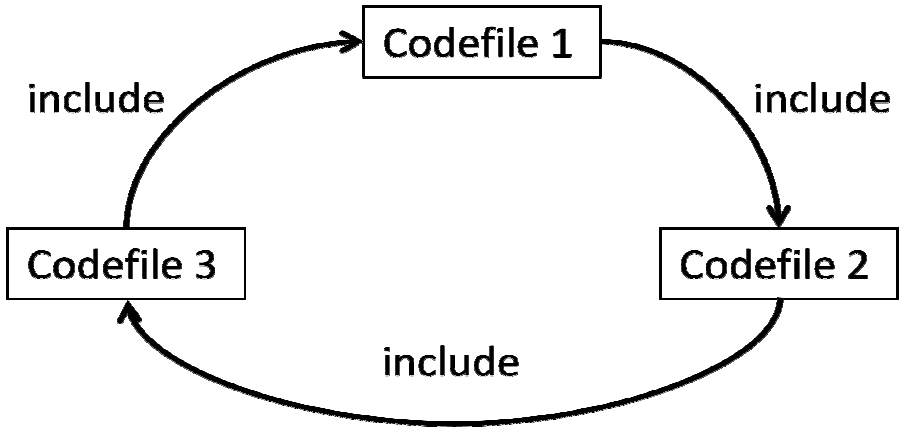


Abbildung 50: Vereinfachte Darstellung Linker-Problem

Der Linker versucht nun während dem Kompilierprozess ausgehend von Codefile 1, Codefile 2 zu Linken, da Codefile 1 auf Codefile 2 angewiesen ist. Jedoch ist Codefile 2 von Codefile 3 abhängig, weshalb der Linker zuerst Codefile 3 linken muss. Hier beginnt das Ganze von vorne, da nun eigentlich Codefile 1 gelinkt werden müsste. Wir haben nun eine Linker-Schleife, der Quellcode ist nicht mehr kompilierbar.

In unserem Projekt kamen wir schlussendlich an einen Punkt, wo wir nicht mehr in der Lage waren, gewisse Klassen zu referenzieren. Im Compiler hatten wir Fehler, wie folgende:

```
d:\projects\fh-bern\squashi3d\lib-i3dengine\_lib-
Equalizer\release\include\eq\fabric\pixelViewport.h(150): error C2059: syntax
error : ')'
```

Lösungsversuch Gesamt-Headerfile

Diese Fehler weisen nicht zwingend auf ein Linker-Problem hin. Durch rechechieren und ausprobieren, haben wir herausgefunden, dass die Reihenfolge von Dritt-Bibliotheken, wie Equalizer, entscheidend ist. Nach Besprechung mit Herrn Künzler wählten wir deshalb den Weg, ein grosses Header File mit allen wichtigen Includes von Dritt-Bibliotheken zu machen. Damit, hofften wir, dass wir eine Reihenfolge herausfinden können, welche so funktioniert.
Damit wir das Testen konnten, musste jedoch der gesamte Source-Code umgeschrieben werden, so dass alle Header dieses Gesamt-Headerfile referenzieren.

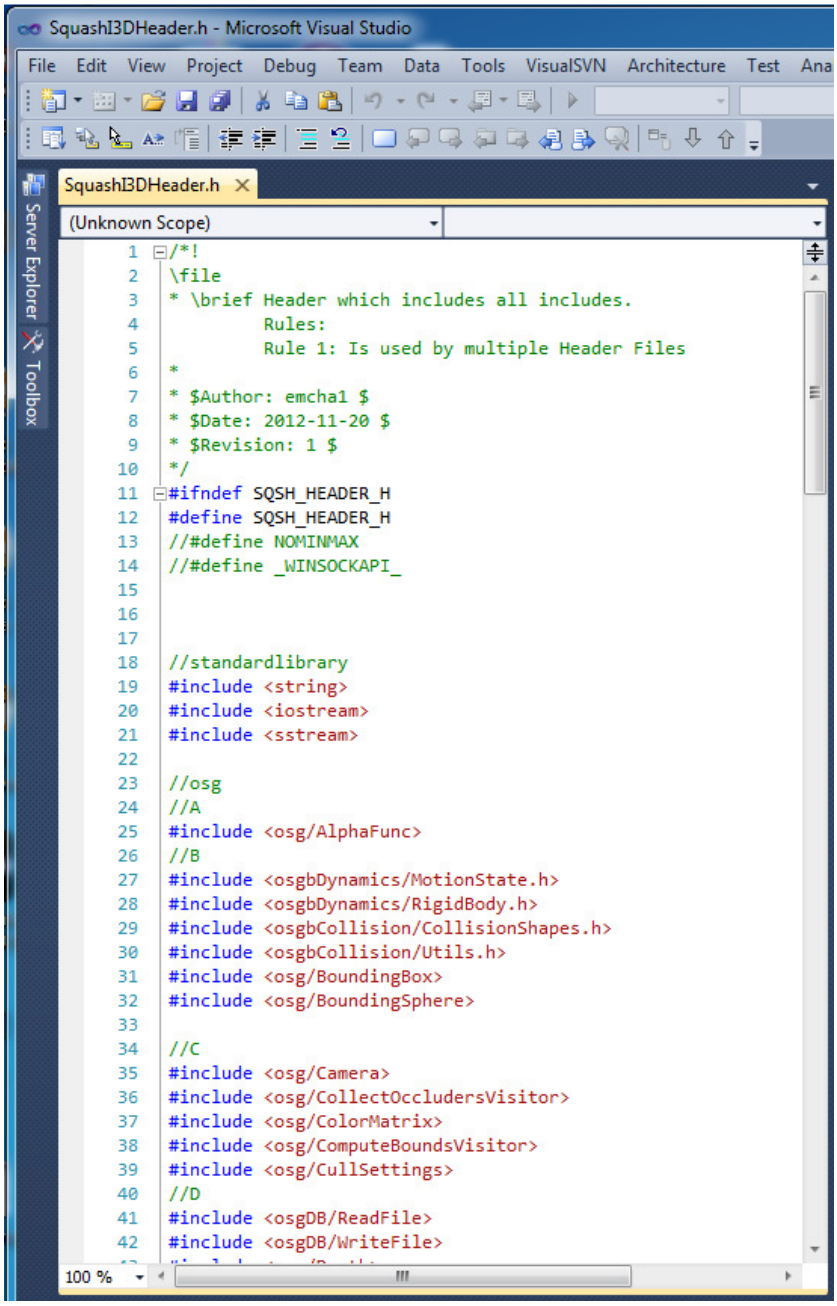


Abbildung 51: Gesamt-Headerfile

Jedoch konnte auch damit unsere Linker-Probleme nicht gelöst werden.

Neuer Lösungsansatz

Da wir nun an einem toten Punkt angekommen waren, mussten wir uns näher mit diesen Includes befassen. Wir fanden heraus, dass die für uns suspekten Forward-Deklaration durch aus ihren Sinn haben.
Mit Forward-Declaration können solche Linker-Schleifen wie in Abbildung 50: Vereinfachte Darstellung Linker-Problem gezeigt, gelöst werden. Die bedeutet, statt eine Includes werden die verwendeten Datentypen „forward declared“. Für den Linker bedeutet dies, er muss sich nicht

mehr um die abhängigen Codefiles kümmern, sondern vertraut darauf, dass diese später schon erstellt werden. Dies birgt natürlich auch ein Risiko.
Wir wenden nun folgende Strategie an:

A und B sind Headerfiles, welche Klassen, Struct, usw. Definitionen enthalten.

Header File Inclusion Regel
- kein Include, falls: A keine Referenz auf B macht
- kein Include, falls: Die einzige Referenz zu B eine „friend“-Deklaration ist
- forward declare B falls: A enthält einen B-Pointer oder Referenz auf B: z.B. B* myb;
- forward declare B falls: eine oder mehrere Funktionen ein B-Objekt/ -Pointer/ -Referenz als Parameter oder Rückgabewert hat: z.B. B MyFunction(B myb);
- #include "b.h" falls: A von B ableitet
- #include "b.h" falls: A ein B Objekt enthält: B myb;

Inclusion Reihenfolge Regel:
Die Includes sollen in folgender Reihenfolge geschehen:
- Mein Projekt-Header
- Header von referenzierten Projekts
- Externe Bibliotheken
- System-Includes

Auch für diese Strategie musste wieder der ganze Source-Code umgeschrieben werden. Jedoch lösten sich damit unsere Probleme und wir mussten uns seither nicht mehr um solche Fehler kümmern.

Die Idee mit dem Gesamt-Headerfile wird in anderen Projekten ebenfalls erfolgreich angewandt. Es nennt sich Precompiled Header. Der Einsatz eines Precompiled Header’s entbindet einem nicht davon, die obengenannte Strategie anzuwenden.
Dieser Fehler kostete uns ca. 60 Arbeitsstunden. Wir verbuchen das als C++-Lehrgeld.

5.6 PotentialFieldForceAlgo und ProxyPointForceAlgo

Das Framework I3D enthält bereits Algorithmen, um haptische Feedbacks zu berechnen. Diese Algorithmen können einfach eingesetzt und angewendet werden.
Bei der Analyse-Phase testeten wir das haptische Feedback mit bestehenden I3D-eigenen Lösungen. Das haptische Feedback wird korrekt zurückgegeben.

Mesh-Objekte als Problemfall

Die I3D-eigenen Lösungen verwenden als Haptic-Pointer (gesteuertes Objekt im Raum) alle eine Kugel. Für den Einsatz in SquashI3D wird jedoch ein Racket (Mesh-Objekt) benötigt.
Mit Erreichen des Meilsteins Ball-Interaktion stand uns ein Prototyp zur Verfügung, um das haptische Feedback zu testen.

Die Testergebnisse in Kurzfassung:
- Das haptische Feedback wird zurückgegeben
- Der Ball ist spürbar
- Der Boden ist spürbar
- Nach wenigen Kollisionen kommt es zu unkontrollierten haptischen Feedbacks. Das Gerät ist nicht mehr zu steuern.

Die Kollisionen wurden zwar relativ gut erkannt, jedoch traten zum Teil Probleme auf, wenn das Mesh bereits zu tief im anderen Objekt war. Die zurückgegebenen Kräfte sind willkürlich und nicht nachvollziehbar. Die Kräfte sind teilweise so gross, dass mit einer Beschädigung des Gerätes gerechnet werden muss.

Kurz Analyse der Algorithmen

Die Algorithmen können nicht in kurzer Zeit analysiert werden. Der Einsatz unseres Rackets in bestehenden I3D-Haptic-Applikation führt zum gleichen Verhalten.
Für einfachere Mesh, wie ein Würfel, konnte das Verhalten ebenfalls nachvollzogen werden, wenn auch nur sehr selten.

Fazit und Entscheid

Wir vermuten, dass der Algorithmus ausschliesslich für Kugel-Pointers gedacht ist. Die genaue Analyse des Algorithmus nimmt mindestens 24 h in Anspruch. Ob anschliessend eine Lösung in absehbarer Zeit zu realisieren ist, ist sehr ungewiss. Diese Risiken sind uns zu gross. Wir verzichten auf das haptische Feedback.

5.7 Einsatz einer Spielerfigur

Im Pflichtenheft vorgesehen ist der Einsatz einer Spielerfigur, welche mit dem Racket verbunden ist. Im Prozess Konzeptphase kamen wir zur Einsicht, dass der Einsatz einer Spielerfigur den Spielspass mindert.

Situationsbeurteilung

	Vorteile	**Nachteile**
Realismus	Zum Racket gehört eine Spielerfigur.	Die Spielerfigur verdeckt die Sicht auf das Feld.
Spielspass	Eine gut-animierte Figur ist ebenfalls ein Spielspass-förderndes Element.	Ist die Spielerfigur minimalistisch animiert, so sieht es eher lächerlich aus und mindert den Spielspass.
Komplexität	-	Der Benutzer muss neben dem haptischen Gerät, welches das Racket steuert, ebenfalls die Spielerfigur bewegen.
Implementation Aufwand	-	Der Implementation für eine gut-animierte Figur ist immense und unrealistisch.

Tabelle 15: Bewertung Spielfigur

Schlussfolgerung

Wir kommen zum Schluss, dass der Einsatz einer Spielerfigur eher den Spielspass mindern, als fördern wird. Eine akzeptable Spielerfigur wäre mit einem unrealistischen Aufwand verbunden. Wir verzichten auf diese Anforderung. Im Spiel wird einzig das Racket, welches direkt durch das Phantom gesteuert wird, angezeigt.

6 Visuelles Design

6.1 Squash-Racket

Am Anfang unserer Arbeit wollten wir eine Spielperson selber darstellen, die mit dem Squash-Racket verbunden ist. So haben wir uns noch ein Model gesucht und die Bones definiert. Relativ schnell haben wir bemerkt, dass dieses Ziel zu hoch gesteckt ist und zu dem den Spielspass nicht erhöhen wird (siehe 5.7 Einsatz einer Spielerfigur). Somit reduzierten wir die "Spielfigur" auf ein Racket, welches frei im Raum verschoben werden kann.
Das Racket haben wir von Hand selber im 3D-Max entworfen. Dazu haben wir ein Squash-Racket als Blueprint[3] heruntergeladen um diese zu modellieren.

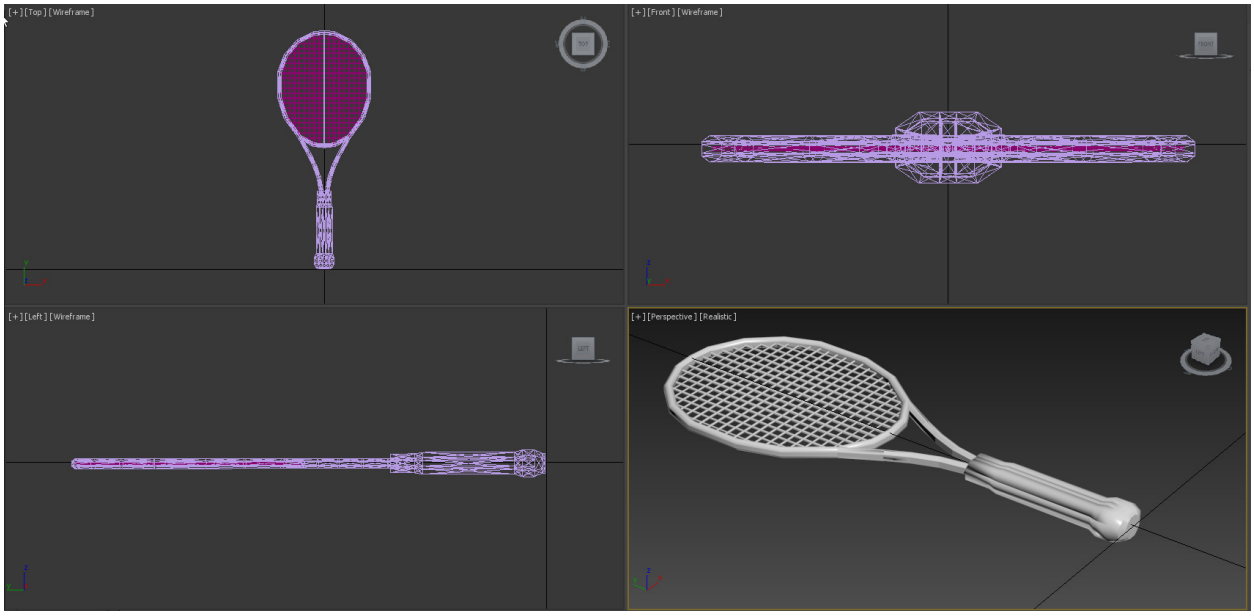


Abbildung 52: Racket im 3D-Max

Zuerst haben wir ein Modell für ein Tennis-Racket heruntergeladen, welches deutlich detaillierter war. Allein die Dateigrösse des Modelles war 1.5 MB gross. Während dem Spiel haben wir deutliche Geschwindigkeitsverluste bemerkt, um die Kollisionen zu berechnen. Deshalb haben wir uns entschieden, ein eigenes Modell zu erstellen, welches einfacher und kleiner in der Dateigrösse ist. Um dies zu erreichen, haben wir bei den Saiten des Rackets eine einfachere Form gewählt, als es bei den Originalen ist. Somit konnten wir die Anzahl Polygone und Vertices klein halten, was die Laufzeit der Algorithmen für die Kollisionsberechnung deutlich verbesserte. Den Unterschied im visuellen war jedoch kaum spürbar, weil die Modelle nie eine Grösse erreichen, bei der solche Details für das Auge relevant werden.

[3] http://en.wikipedia.org/wiki/Blueprint

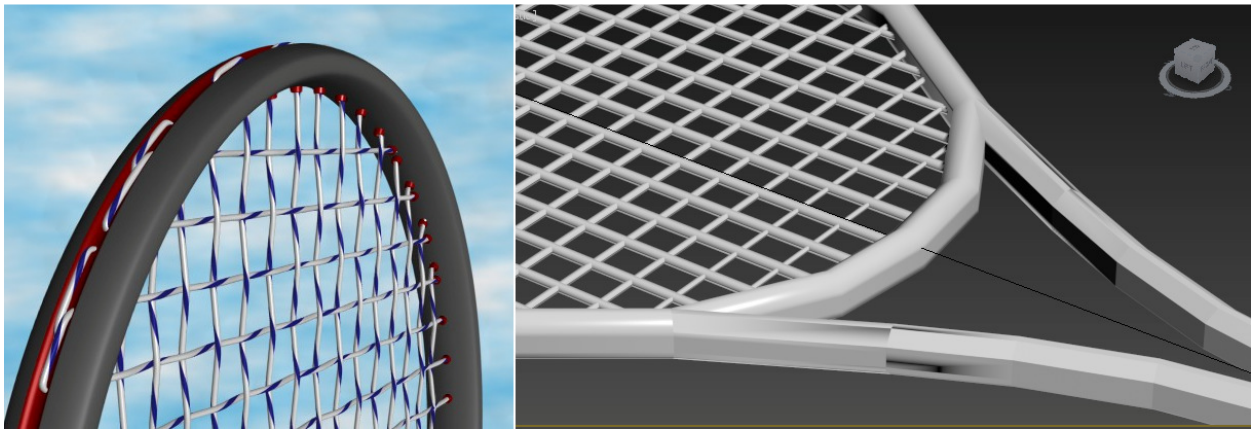


Abbildung 53: Vergleich Original und Modell

Nach vielen Tests haben wir regelmässig bemerkt, dass die Frame-Rate einsank, sobald eine Kollision eingetreten ist. Sobald die Kollisionen zwischen zwei Rackets stattfand, blieb das Spiel sogar für einige Sekunden komplett stehen, weil für die verschiedenen Kollisionspunkte zu viel gerechnet werden musste. Durch dieses Verhalten haben wir uns entschieden, die Möglichkeit einzubauen, bei der ein Mesh für das Grafische und eines für die Physik geladen werden können. Somit mussten wir ein zweites Mesh für das Racket erstellen, welches für ein CollisionShape in Bullet optimiert ist. Dazu haben wir dieselbe Form des Rackets wie bisher genommen, jedoch die Form noch einmal sehr vereinfacht. So sind die Saiten jetzt eine eigene Fläche und ähnelt mehr einem Ping-Pong Schläger. Dadurch konnten die Reaktionszeiten bei Kollisionen weiter verringert werden.
Durch diesen Ansatz ist SquashI3D auch für Änderungen in der Zukunft relativ flexibel und kann für grössere und komplexere Modelle angewendet werden. Im Hintergrund wird mit einem entsprechend einfacheren Mesh für die Kollisionserkennung gearbeitet. Eine weitere Möglichkeit wäre ebenfalls gewesen, die Vereinfachung des Meshs automatisch durch OSG mit "osgUtil::Simplifier" zu realisieren. Dazu muss ein Prozentsatz angegeben werden, um welchen das Mesh vereinfacht werden soll. Wie das Mesh schlussendlich vereinfacht wird, liegt aber nicht in den Händen des Benutzers. Dies kann ein Nachteil sein bei verschiedenen Modellen. Bei unserem Racket hat es zum Beispiel ein paar Saiten heraus gelöscht. Um für die Kollisionen ein adäquates Model zu erhalten, muss die Verantwortung beim Benutzer liegen und dieser kann ein entsprechendes Model dazu laden.
Die Vereinfachung von OSG ist vor allem dazu gedacht, diese für sogenannte LOD-Algorithmen zu verwenden. LOD steht für Level-Of-Detail und ersetzt je nach Distanz des Models zur Kamera das Modell. So kann bei einer weiten Distanz ein stark vereinfachtes Model verwendet werden, was die Rechenzeit nochmals verkleinert. Bei einer Kollisionserkennung mit einem Ball im Squash sollte jedoch ein sauberes ColliisonShape gewährleistet sein.

Unser vereinfachtes CollsionShape für Bullet sieht im 3D-Max wie folgt aus:

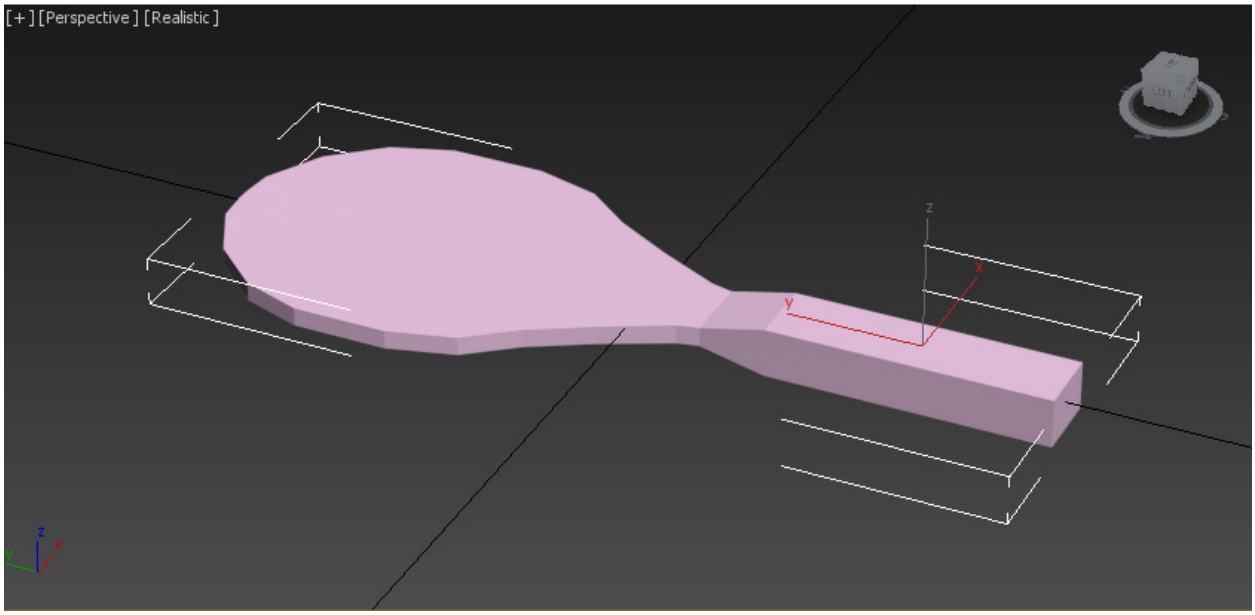


Abbildung 54: Mesh für das CollisionShape für Bullet

6.2 Squash-Halle

Um unser Squash-Spiel grafisch interessant und schön zu gestalten, haben wir uns entschieden ein Squash-Court darzustellen, wie es an Weltmeisterschaften zu sehen ist. Im Pflichtenheft hatten wir beschrieben, dass wir nur einen Court von innen darstellen, ohne das eine Umgebung wahrgenommen werden kann.

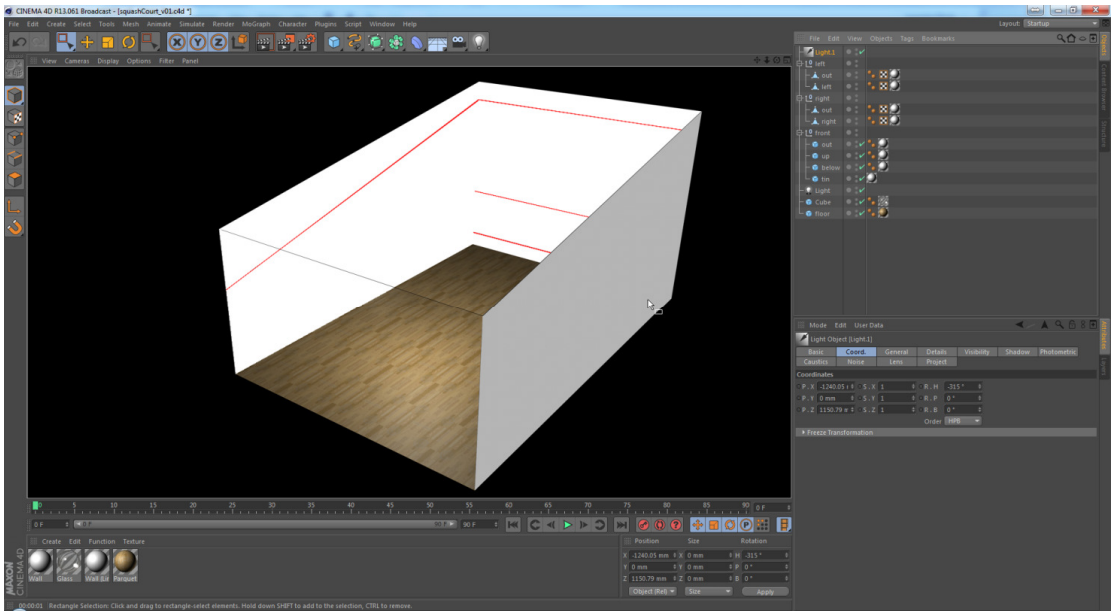


Abbildung 55: Modell-Prototyp aus dem Pflichten-Heft.

Während der Arbeit haben wir ein 3D-Model einer Squash-Halle gefunden, welche wir im 3D-Max für unsere Zwecke bearbeitet und verändert haben. Dieses Modell umfasst nur die Halle. Das Court (ähnlich wie bei Abbildung 55: Modell-Prototyp aus dem Pflichten-Heft.) selber wird durch den Level-Creator mit Hilfe des XML erstellt und in die Halle geladen.
Die Halle sieht im 3D-Max wie folgt aus:

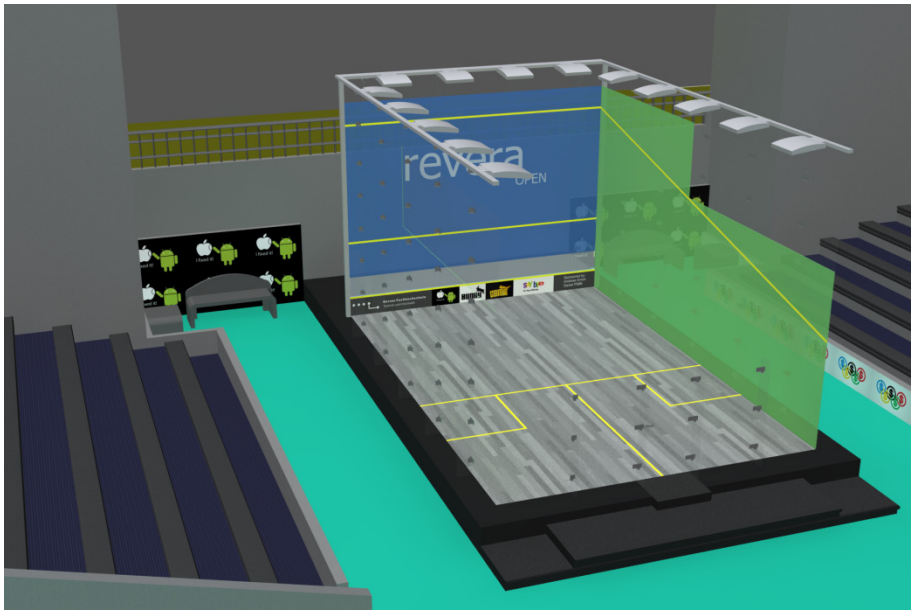


Abbildung 56: Squash-Halle im 3D-Max

6.3 Schatten

Damit sich der Spieler im Spiel besser orientieren kann, haben wir der Scene einen Schatten bezüglich einer in XML-definierten Lichtquelle hinzugefügt. Dazu haben wir die ShadowedScene von der osgShadow-Library dem OSG-Root-Knoten des SceneManagers hinzugefügt. Alle OSG-Knoten, welche entweder einen Schattenwurf erzeugen, erhalten oder beides, müssen unter dieser ShadowedScene angehängt werden. Mit unserer Implementation ist nur eine Lichtquelle unterstützt. Die weiteren werden keinen Einfluss auf den Schattenwurf haben.

Durch diese kleine Änderung am SceneGraph konnten wir erreichen, dass die Navigation im 3D-Raum erheblich besser und einfacher wurde. Es ist einfacher sichtbar, in welchem Verhältnis der Schläger zum Ball steht und wo sich beides im Raum befindet.

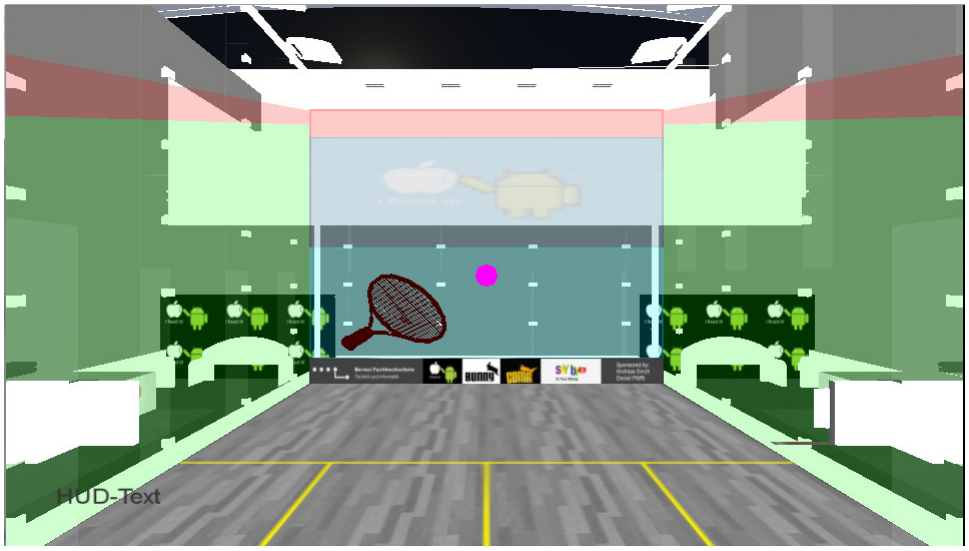


Abbildung 57: Spiel ohne Schatten

In der Abbildung oben ist unklar, ob sich der Ball hinter oder vor dem Racket befindet.

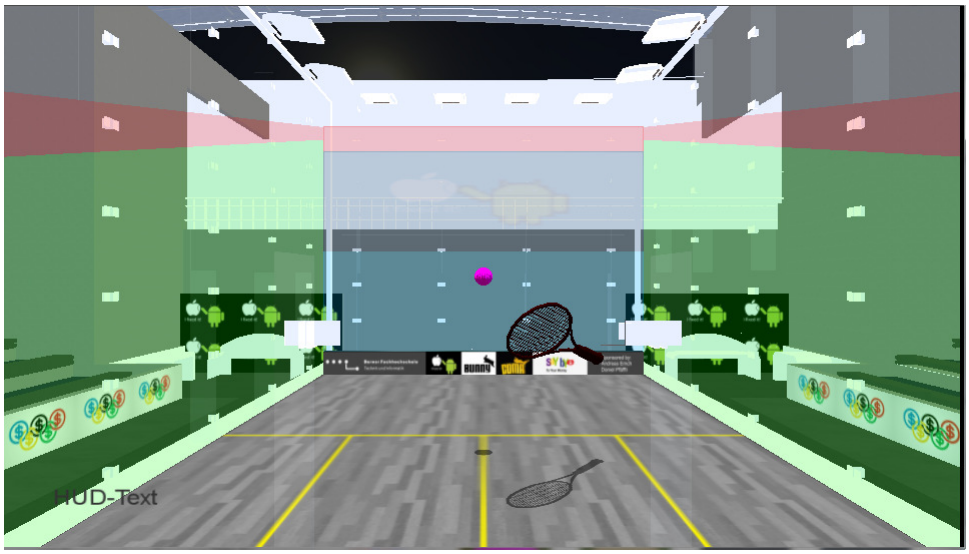


Abbildung 58: Spiel mit Schatten

Das Gefühl für die Tiefe ist sofort da. Der Ball ist klar vor dem Racket.

6.4 Shader

Um das Menu ansprechender zu gestalten und nicht nur eine schwarze Leere am unteren Rand zu erhalten, haben wir einen Wasser-Shader eingesetzt. Das Menu, die Rackets sowie der Sternenhimmel spiegelt sich darin und das Menu ist visuell ansprechender.

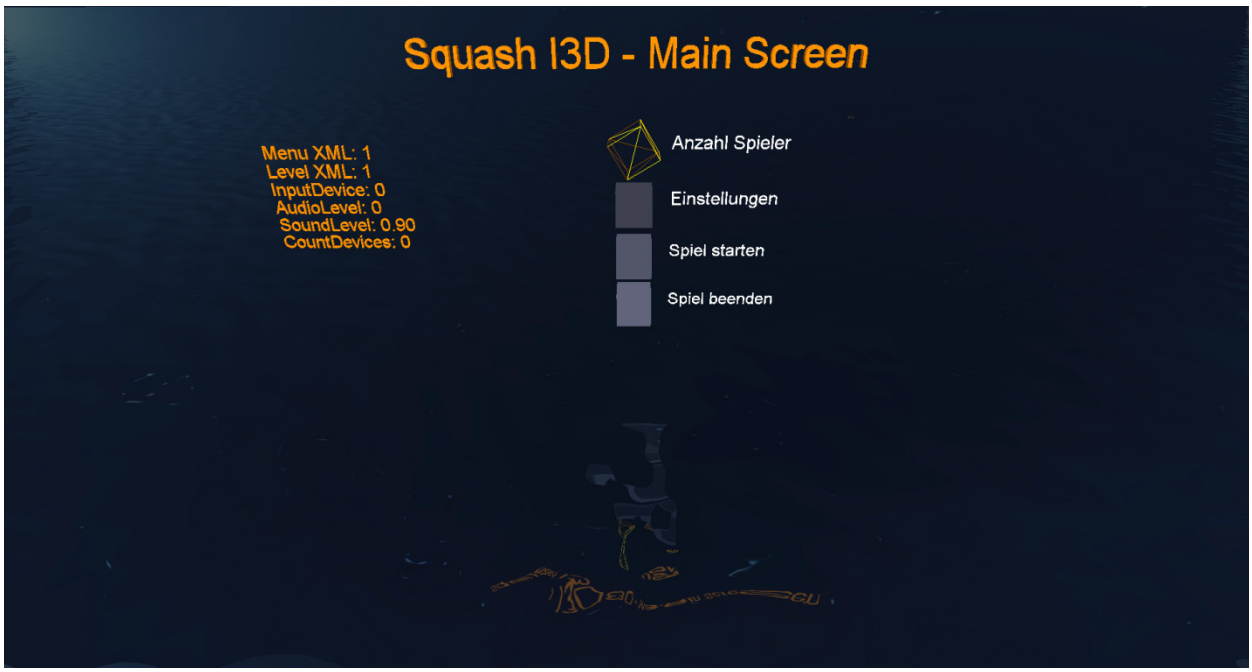


Abbildung 59: Das Menu mit einem Wasser-Shader mit Reflektion

7 Benutzerhandbuch

SquashI3D ist ein Spiel, welches für den Einsatz im BFH-CAVE konzipiert wurde. Es kann jedoch ebenfalls auf einem normalen Computer gespielt werden, wobei da auf den 3D-Effekt verzichtet werden muss. SquashI3D wurde im Rahmen einer Bachelor-Thesis realisiert.

Systemvoraussetzungen

Folgend ein empfohlenes System:
- Intel Core i7-2820 2..30GHz
- Nvidia Quadro 3000M
- OCZ-Vertex 2
- 4 GB RAM
- SensAble Phantom Omni

Der Einsatz einer SSD wird empfohlen, da grössere Datenmengen zur Laufzeit geladen werden müssen. Ohne SSD wird dieser Vorgang einige Zeit in Anspruch nehmen.
Wir weisen ausdrücklich darauf hin, dass diese Applikation unter AMD-Grafikkarten nicht flüssig zu benutzen ist (0-3 FPS). Dies da Dritt-Komponenten Optimierungen zu Gunsten von Nvidia gemacht haben.
SquashI3D ist unspielbar ohne SensAble Phantom Omni.
64-Bit ist zwingend.

7.1 Spielprinzip

SquashI3D kennt zwei Spieler:
- Spieler Rot
- Spieler Grün

Die Farbe kennzeichnet ebenfalls das Racket.

Aufschlag:

Der Aufschlag wird immer aus einem der beiden Aufschlagfelder ausgeführt. Der Spieler Rot schlägt immer aus dem Feld Rechts auf. Der Spieler Grün aus dem Feld Links.
Nach dem Aufschlag muss der Ball die Rückwand oberhalb der Werbelinie treffen. Wenn der Aufschlagende den Ballwechsel verliert, erhält der Gegenspieler das Aufschlagrecht. Im Gegensatz zu anderen Racket-Sportarten darf der Ball beim Squash für den Aufschlag auch mit dem Schläger angeworfen werden.

Ballwechsel

Der Ball muss nach jedem Schlag auf direktem oder indirektem Weg die Vorderwand berühren. Als indirekt gilt ein Weg über Seiten- und Rückwand. Danach darf der Ball nicht mehr als einmal auf dem Boden, jedoch beliebig oft auf die Seitenwände auftreffen, bevor er vom Spielpartner zurückgeschlagen wird. Ein Ball gilt im „Aus“, wenn er die Wände oberhalb der dort angebrachten roten Begrenzungslinien, die Begrenzungslinie selbst oder das Tin berührt.

Zählweise

Es gilt, dass ein Spiel über beliebig viele Sätze gehen kann, d. h. SquashI3D begrenzt die Anzahl Sätze nicht. Es wird jeder Punkt gezählt, egal wer das Aufschlagrecht hatte. Für den normalen Satzgewinn benötigt der Spieler 21 Punkte. Beim Stand von 21:20 wird ein Tie-Break gespielt. Hier gewinnt derjenige Spieler, welcher zuerst 2 Punkte Vorsprung hat (z. B. 23:21, 27:25 usw.).

7.2 Virtuelle Realität

SquashI3D hat keinen Anspruch eine Simulation der Sportart Squash zu sein. Dennoch nachfolgend ein Vergleich von Realität und Spiel.

	SquashI3D	**Realität**
Ballgeschwindkeit (max.)	begrenzt auf 80 km/h	bis zu 200 km/h
Spielfeld Länge	9.75 m	9.75 m
Spielfeld Breite	6.40 m	6.40 m
Racket Kopfgrösse (Ø)	4.0 m^2	ca. 0.5 m^2
Durchmesser Ball	100 mm	39.0 – 40.5 mm

Tabelle 16: Virtuelle Realität vs. Realität

7.3 Tastenkombination

In SquashI3D können alle Aktionen mit Hilfe des SensAble Phantom Omni gesteuert werden. Es wird keine Tastatur benötigt. Folgende Tasten sind belegt:

Taste	**Aktion**
s	Zeigt die Statistik zu Debugzwecken an.
W/w	Selektiert im Menu den vorderen Punkt (oberhalb).
E/e	Selektiert im Menu den nächsten Punkt (unterhalb)
Q/q/<Enter>	Wählt im Menu die aktuelle Selektion aus.
P/p	Unterbricht das Spiel oder setzt es fort.
D/d	Aktiviert den Debug-Mode von Bullet.
F1	Wechselt ins Menu
F2	Startet das Spiel

Tabelle 17: Tastaturbelegung

7.4 Haptic-Steuerung

Das SensAble Phantom Omni kennt nur zwei Tasten. Je nachdem, ob das Menu aktiv ist, oder das Spiel, stehen die beiden Tasten für unterschiedliche Aktionen.
Um ein Menu-Punkt auszuwählen, muss das Racket, gesteuert durch das Phantom Omni, auf den Menu-Punkt zugesteuert werden. Trifft das Racket auf einen Menu-Punkt, wird dieser im Menu ausgewählt (siehe Abbildung 60: Ausgewählter Menu-Punkt). Mit dem Button1 (blauer Knopf) kann der ausgewählte Menu-Punkt ausgeführt werden.

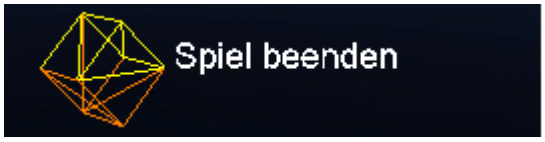


Abbildung 60: Ausgewählter Menu-Punkt

Die Steuerung im Menu bleibt dem Spieler Rot vorbehalten (rotes Racket).

Tastenbezeichnung und -belegung

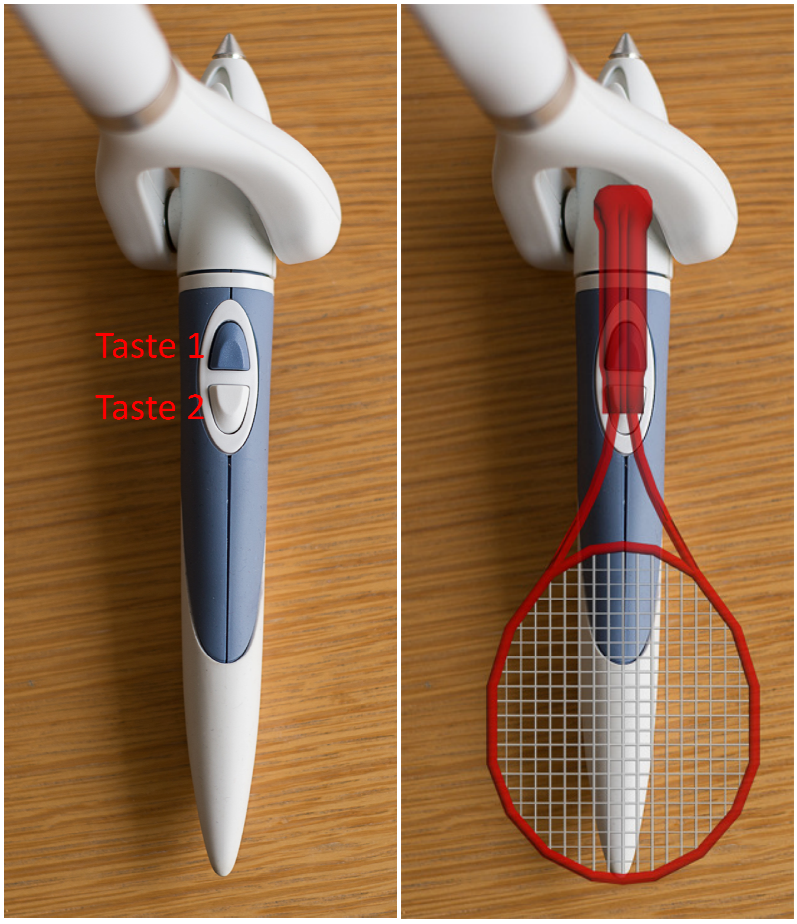


Abbildung 61: Tasten Phantom Omni

Das Racket wird, wie oben gezeigt, gesteuert. Das bedeutet man schlägt nicht mit dem Stift, sondern mit dem gegenüberliegenden Ende.
Die dunklere, der beiden Tasten, bezeichnen wir als Taste 1. Die andere Taste 2.

Taste	Menu-Aktiv	Spiel-Aktiv
Taste 1	Wählt den aktiven Menu-Punkt aus.	-
Taste 2	-	Setzt den Ball zurück.
Taste 1 + 2	-	Wechselt ins Menu.

Tabelle 18: Tastenbelegung Phantom Omni

7.5 Installation

7.5.1 Kompilieren der Source

Das Kompilieren der Sources ist nicht zwingend erforderlich. In der Beigelegten DVD steht eine kompilierte 64Bit-Version von SquashI3D zur Verfügung.
Unser Projekt kann mit Hilfe von CMake auf allen verschiedenen Windows, Linux und Mac Systemen kompiliert werden. Dazu kann mit Hilfe des Source-Ordners ein Programmierprojekt für die gewünschte IDE erstellt und damit kompiliert werden. Wir haben jeweils nur mit Windows 7 & Windows 8 für die IDE Visual Studio 2010 64Bit gearbeitet.

7.5.2 Einrichten der Haptic

In der DVD beigelegt ist die Treiber Datei von SensAble Phantom Omni „/drivers/ Phantom_Device_Drivers_5.1.7_Release.exe“. Führen Sie die Datei aus und folgen Sie den Anweisungen des Setups. Starten Sie anschliessend das System neu.

Das Phantom Omni Gerät muss zwingend mit folgenden Einstellungen eingerichtet werden. Andere Einstellungen können dazu führen, dass unsere Applikation keines oder nur ein Phantom erkennen kann.

In dem Konfigurations-Programm, welches mit den Treibern von SensAble mit geliefert wird, können die angeschlossenen Phantom Omni Geräte eingerichtet werden. Aufgrund der Einschränkung, dass mit Chai3D nur maximal zwei Geräte verwalten werden können, sollten für dieses Beispiel nur zwei Geräte angeschlossen und eingerichtet werden. Um Korrekt erkannt zu werden, müssen die beiden Geräte wie folgt beschrieben werden:

1. Gerät: Der Name muss "Default PHANToM" oder "Phantom1" sein. Die Seriennummer kann frei gewählt werden.
2. Gerät: Der Name muss "Phantom2" sein. Für dieses Gerät muss die andere Seriennummer eingestellt werden.

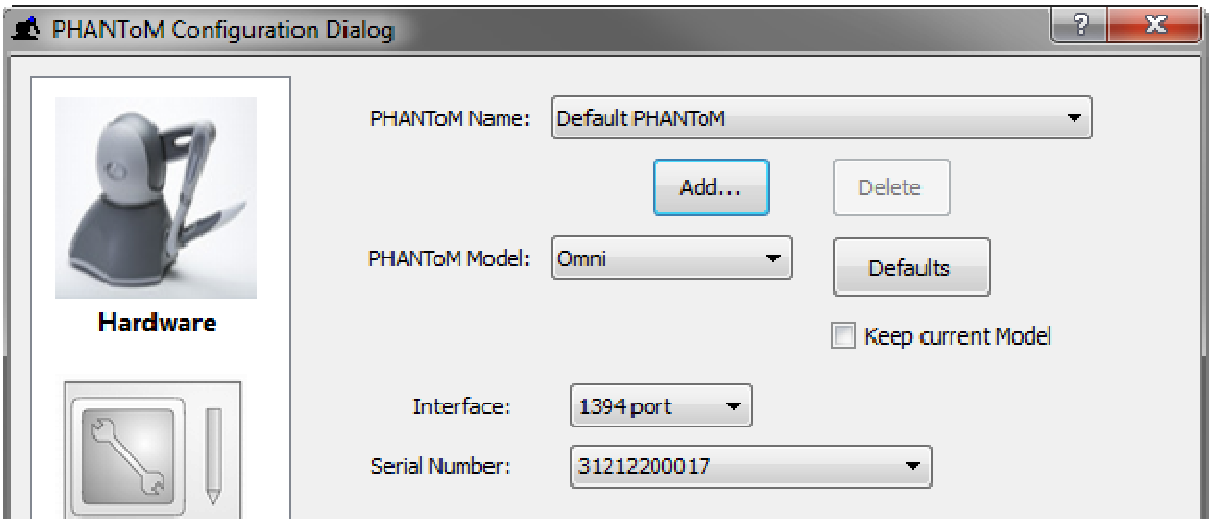


Abbildung 62: Einstellungen Gerät 1

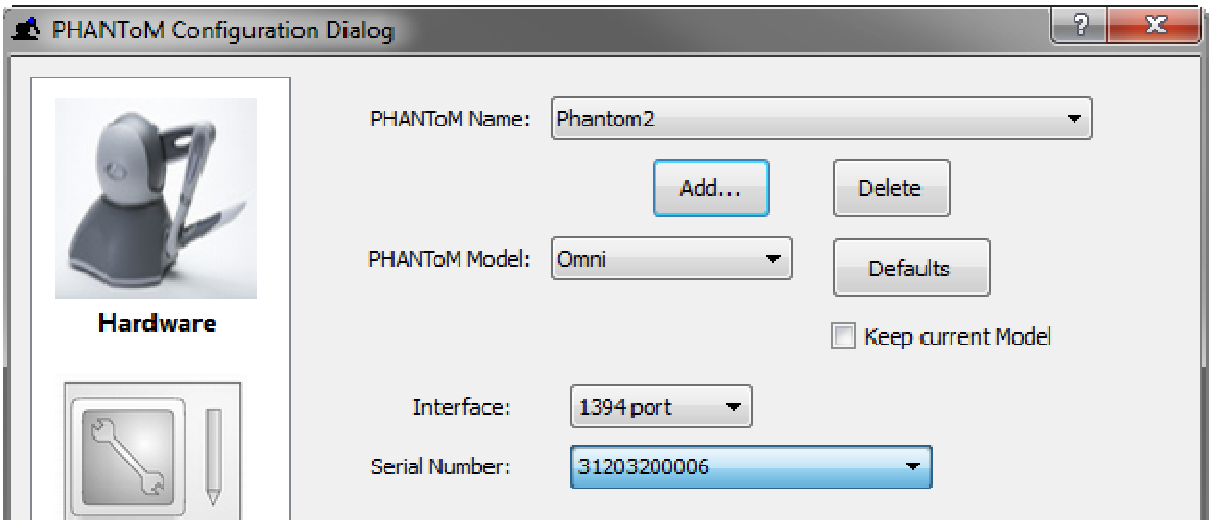


Abbildung 63: Einstellungen Gerät 2

7.5.3 Starten des Spiel auf einem PC

In der DVD enthalten ist ein Ordner „/SquashI3D“ enthalten. Kopieren Sie diesen Ordner auf einen lokalen Datenträger.
Im Ordner enthalten ist die Datei „/SquashI3D/ SquashI3D.exe“. Dies ist die Start-Datei von SquashI3D. Führen Sie diese aus. Die Applikation wird gestartet.

Das Spiel beginnt im Squash-Raum. Bitte warten Sie bis das Menu angezeigt wird. Im Menu angekommen, stellen Sie die Anzahl Spieler ein. Das haptische Gerät wird erst initialisiert, wenn Sie die Anzahl Spieler angegeben haben. Die Geräte sind korrekt konfiguriert, sobald Sie mit Hilfe des Gerätes ein Racket steuern können.

SquashI3D ist ausgelegt, um mit haptischen Geräten gespielt zu werden. Mit der Tastatur kann zwar das Menu gesteuert werden, jedoch nicht das Spiel.

7.5.4 Starten des Spiels im CAVE

Die kompilierten Dateien müssen auf allen Computer im CAVE die gleichen sein. Sofern das Programm im Quellcode verändert und neu kompiliert wurde, müssen diese Dateien auf allen Computer aktualisiert werden. Dies kann über eine Bat-Datei auf dem gemacht werden. Die Datei kopiert alle Dateien vom Squash-I3D, welche im Verzeichnis "D:\CaveApps\SquashI3D" liegen müssen, auf die Client-Nodes.

Nach dem Aktualisieren kann das Spiel mithilfe einer anderen BAT-Datei über den Equalizer im CAVE gestartet werden (siehe Equalizer-Dokumentation für genauere Einstellungsmöglichkeiten). Diese startet das Spiel auf allen Clients in dem Client-Modus und am Schluss das Spiel im Server-Modus auf dem Master. Nach dem Ladevorgang kann das Spiel wie auf dem normalen Computer verwendet werden.

SquashI3D ist ausgelegt, mit nur einem Haptic-Computer zu arbeiten. Dies bedeutet, alle Haptic-Geräte müssen an einem Computer angeschlossen sein. Der Einsatz von zwei Haptic-Computer ist nicht vorgesehen. Die Haptic-Geräte müssen nicht zwingend am Master angeschlossen sein.

8 Testing

8.1 UnitTests

SquashI3D ist eine graphische Applikation. Das bedeutet, dass wir nur sehr wenig sogenannte Businesslogik haben. Die Businesslogik ist bei uns in der BallStateMachine enthalten und damit getrennt vom Rest. Wir haben uns entschlossen die Frameworks, wie Bullet, Chai3D, OSG nicht mit UnitTests zu testen. Das bedeutet, wir vertrauen darauf, dass diese Frameworks bereits getestet wurden.
Unser Testing beschränkt sich auf folgende Bereiche:

- BallStateMachine
- Serialize/ Deserialize Event-System
- CollisionManager

Das Testing haben wir ohne UnitTest-Framework umgesetzt, da der Einsatz eines solchen noch einmal einen zusätzlichen Mehraufwand bedeutet hätte. Da wir unter 50 Testcases haben, schien sich für uns den Aufwand nicht zu lohnen.

Die TestCases werden mit Hilfe einer Compiler-Directive in der main.cpp Datei eingeschaltet.

Ein TestCase ist in SquashI3D nach folgenden Regeln aufgebaut:

- Der Rückgabewert gibt an, ob der Testlauf erfolgreich war. (true/false)
- Der Test schreibt in die Console, dass er gestartet wurde
- Der Test schreibt in die Console, ob der Lauf erfolgreich war.

```
bool TestCase_GameSettingsChangedEvent_Case2()
{
    I3D_LOG(notification) << "Run TestCase_GameSettingsChangedEvent_Case2" << std::endl;
    SQUASHI3D::EventSystem::Events::GameSettingsChangedEvent* gamesettingschangedevent =
        new SQUASHI3D::EventSystem::Events::GameSettingsChangedEvent();

    bool b = gamesettingschangedevent->hasChanges()==false;
    if(b)
    {
        I3D_LOG(notification) << "TestCase_GameSettingsChangedEvent_Case2: Test successfull" << std::endl;
    }else{
        I3D_LOG(notification) << "TestCase_GameSettingsChangedEvent_Case2: Test failed" << std::endl;
    }
    return b;
}
```

Abbildung 64: Beispiel TestCase

Schlägt ein Test fehl, werden die anderen Tests nicht mehr ausgeführt.

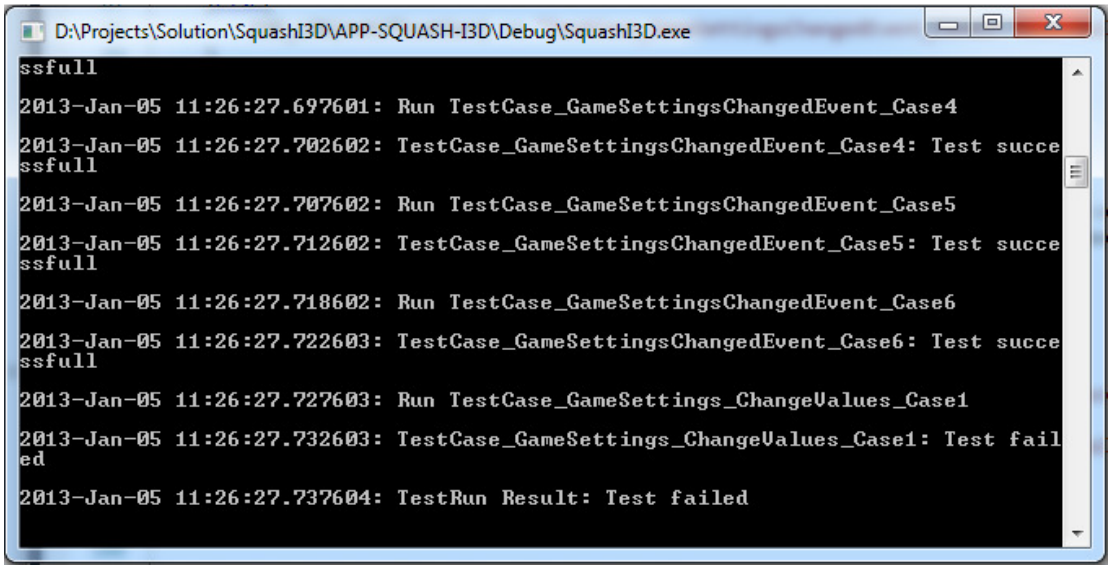


Abbildung 65: Fehlgeschlagener Testlauf

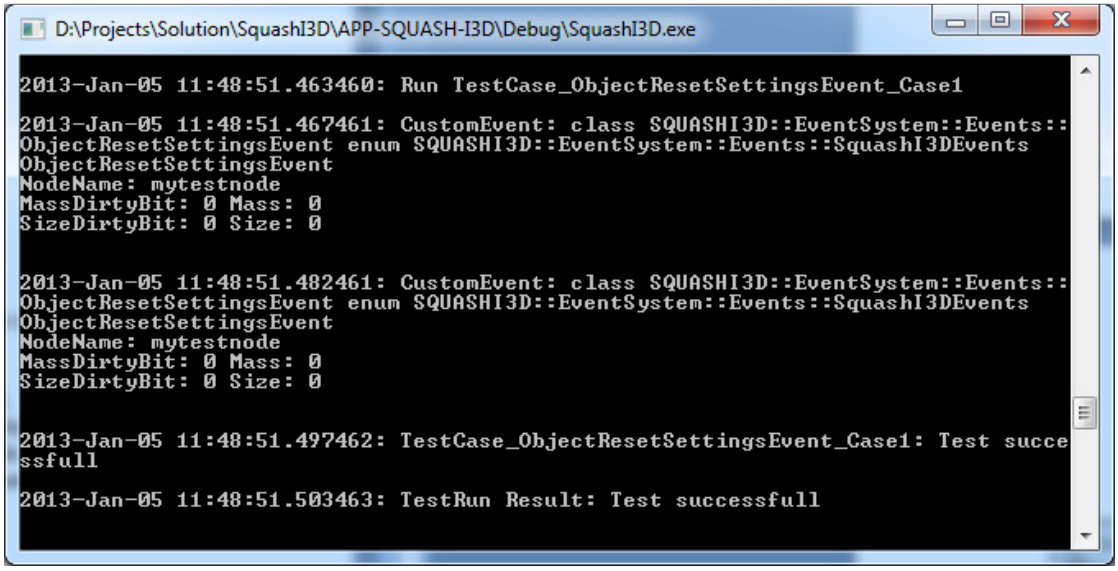


Abbildung 66: Erfolgreicher Testlauf

Es existieren folgende Testfälle:

- Test_CustomEvent
- Test_GameSettingsChangedEvent
- Test_GameSettings
- Test_CollisionManager
- Test_StateMachine
- Test_ObjectChangedEvent
- Test_MenuEvent
- Test_ObjectResetSettingsEvent
- Test_ObjectBlockEvent

Pro Test existieren mehrere Testfälle, welche jeweils einen speziellen Aspekt der Tests anschauen.

8.2 Test-Cases anhand Use-Cases

UseCases sind für ein Spiel schwierig zu finden. Für Computerspiele erstellt man im Normalfall ein Storyboard. Für unser Squash ist ein Storyboard durch die entsprechenden Squash-Regeln gegeben. Anhand von diesen Regeln haben wir bereits im Pflichtenheft unter dem Kapitel "6.12 Use-Cases" verschiedene Use-Cases definiert. Weil sich diese auch gut als Test-Cases eignen, haben wir diese als Test-Cases übernommen. Folgend sind alle Cases erläutert:

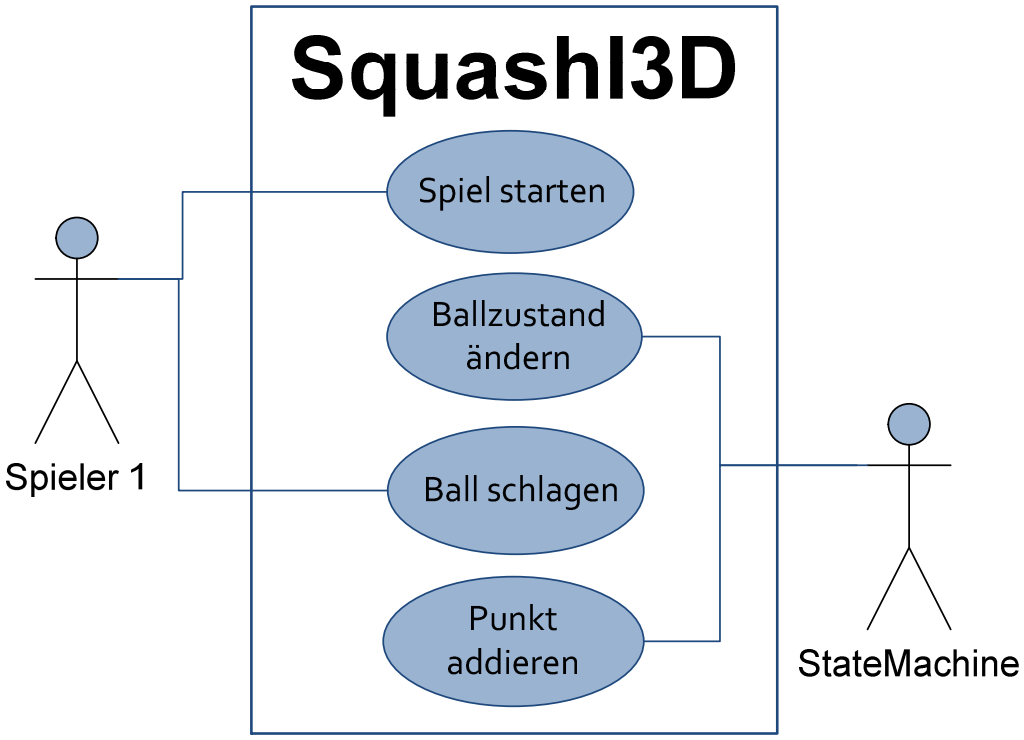


Abbildung 67: Übersicht Test-Cases

8.2.1 Spiel starten

Nach dem Starten der BAT-Datei, welche zuständig ist, um die Anwendung im CAVE zu starten, hat der Spieler im CAVE die Möglichkeit, verschiedene Einstellungen gemäss zu ändern und/oder das Spiel zu starten.

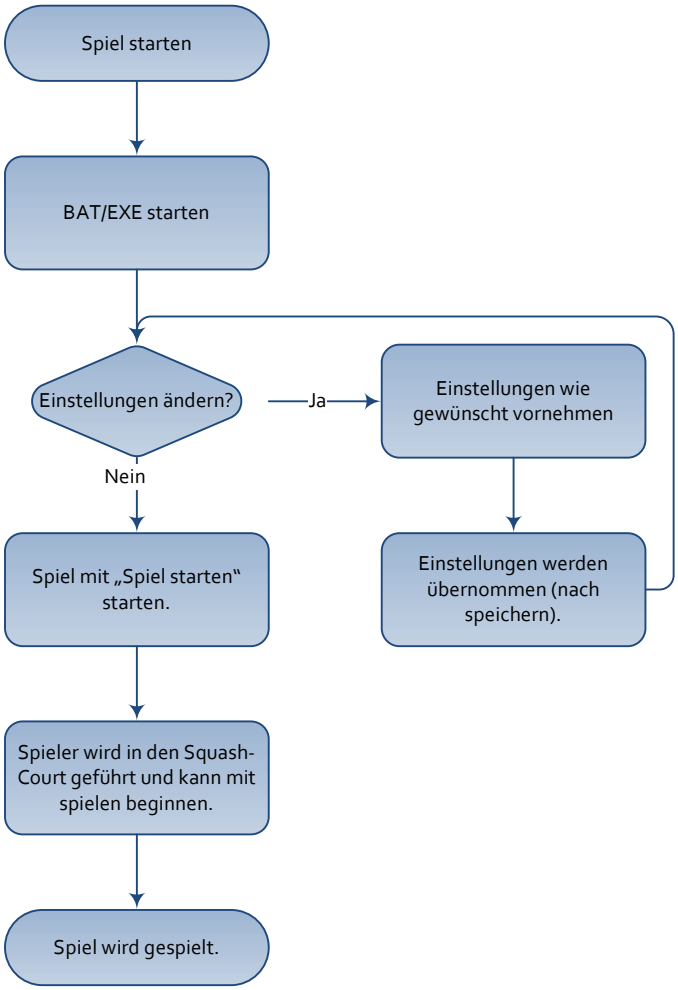


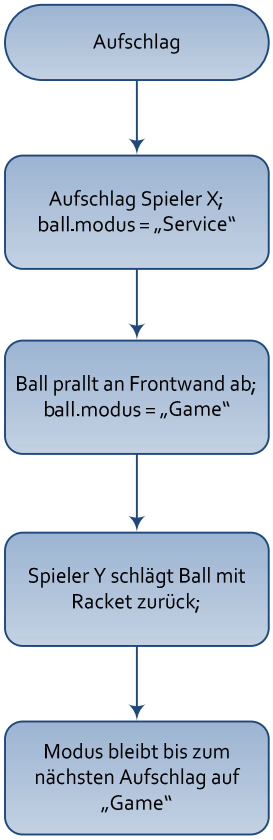
Abbildung 68: Test-Case – Spiel starten inkl. Einstellungen ändern.

Tests

Die verschiedenen Einstellungen können sowohl im CAVE, wie auch am Personal Computer vorgenommen werden. Diese werden direkt übernommen und in den GameSettings gespeichert. Anhand der Einstellungen wird die Musik abgespielt und das richtige Level geladen, sobald das Spiel gestartet wird.

8.2.2 Ball-Zustand ändern

Der Aufschlag wird im Squash als einen Spezialfall von den üblichen Ballwechseln gesehen. Beim Aufschlag muss der Ball oberhalb der Aufschlaglinie der Frontwand abprallen. Ansonsten ist es ein ungültiger Aufschlag. Um dies möglichst einfach realisieren zu können, wird der Ball mit einem Modus-Attribut versehen, um beim Aufprall an der Frontwand den entsprechenden Fall überprüfen zu können.



Mögliche Ballmodi: „Service" und „Game". Somit kann während dem Ballwechsel aufgrund dieses Modus die entsprechende Regel angewendet werden.

```
X   = 0|1
Y   = !X
```

Abbildung 69: Test-Case – Ballmodus

Tests

Der Modus des Balles, ob dieser im Aufschlag ist oder nicht, wird über ein Attribut der BallStateMachine abgebildet. Dieses wird entsprechend der Zustandsänderungen richtig übernommen. Sobald der Status der BallStateMachine auf Start gesetzt wird, wird das Attribut auf true (im Aufschlag) gesetzt. Sobald der Ball an der Frontwand abprallt, wird dieses auf False gesetzt. Diese Änderungen werden gemäss verschiedenen Tests beim Spiel korrekt übernommen. Zusätzlich konnte dieser Case programmiert simuliert werden, indem die gewollten Zustandsänderungen an die BallStateMachine gesendet werden und die Attribute überprüft werden.

8.2.3 Ball schlagen

Der Ballwechsel betrifft folgende Zeitspanne: Spieler X schlägt mit dem Racket den Ball bis zum Zeitpunkt, wo der Spieler Y diesen mit seinem Racket abnimmt. Anschliessend beginnt ein neuer Ballwechsel.
Der Ballwechsel wird beendet, sobald ein ungültiger Fall eintritt, welcher die Regeln bricht. Sobald der Ballwechsel abgebrochen wird, muss die BallStateMachine dem korrekten Spieler den Punkt addieren.

Das Diagramm zum Test-Case ist auf der nächsten Seite abgebildet.

Tests

Der Ballwechsel haben wir zu zweit mit zwei Haptic-Geräten einige Male simuliert, um die Korrektheit der Zustandsübergänge in der BallStateMachine zu überprüfen. Zusätzlich konnten wir mit einem programmierten Test die BallStateMachine prüfen, indem wir gezielte Statuswechsel übergaben und das gelieferte Resultat mit dem Erwarteten verglichen.

Während den Tests ist uns aufgefallen, dass nicht alle Kollisonen des Balles korrekt an die BallStateMachine überliefert werden. Dies ist leider ein Problem der Kollektionserkennung mit Bullet, und nicht in der BallStateMachine selber. Wir konnten nur feststellen, dass Kollisionen nicht an den CollisionManager übertragen wurden. Alle Kollisionen, die an den CollisionManager übergeben werden, sind aber korrekt erkannte Kollisionen. Für die Bewegung des Balles werden die Kollisionen in Bullet selber jedoch richtig verwendet. Das Resultat im Spiel ist dann, dass nicht alle Fälle abgefangen werden, wenn eine Regel missachtet wird. Es kann zum Beispiel vorkommen, dass der Ball erst bei der dritten Kollision mit dem Boden als Fehlspiel erkannt wird anstatt bereits beim zweiten Mal. In diesem Fall wurde entweder die erste oder die zweite Kollision nicht korrekt an die BallStateMachine übermittelt.
Wir können dies mit den programmierten Tests bestätigen, weil wir bei diesen immer das korrekte Resultat erhielten.

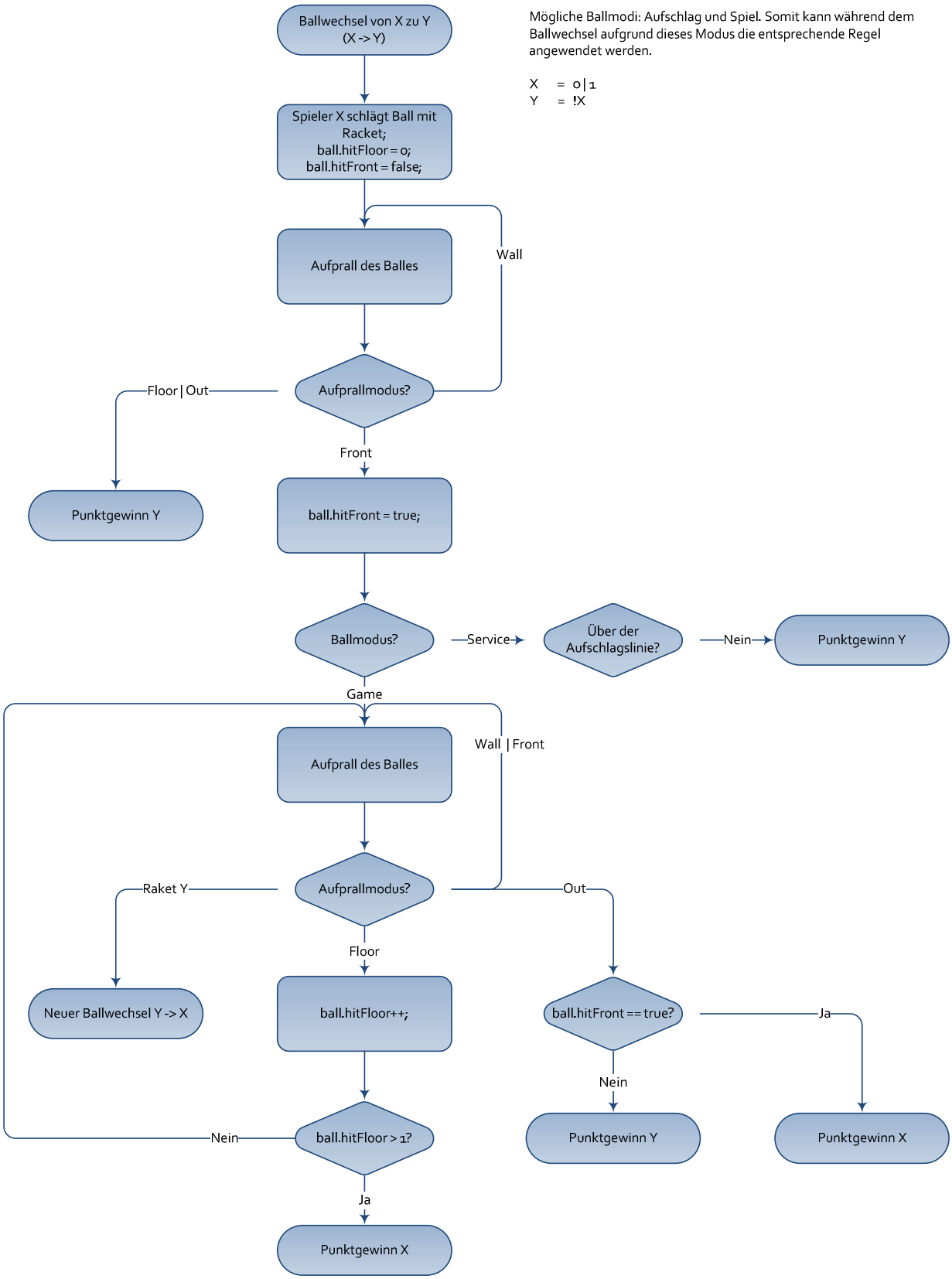


Abbildung 70: Test-Case – Ballwechsel

8.2.4 Punkte addieren

Ziel eines jeden Spielers ist es, als Erster die benötigte Punktzahl zu erreichen. Der erste Spieler, welcher diese erreicht, gewinnt den Satz und in unserem Spiel auch gleich das gesamte Spiel. In diesem Use-Case wird definiert, wann ein solcher Gewinner festgelegt wird. Startpunkt des UseCases ist ein Punktegewinn.

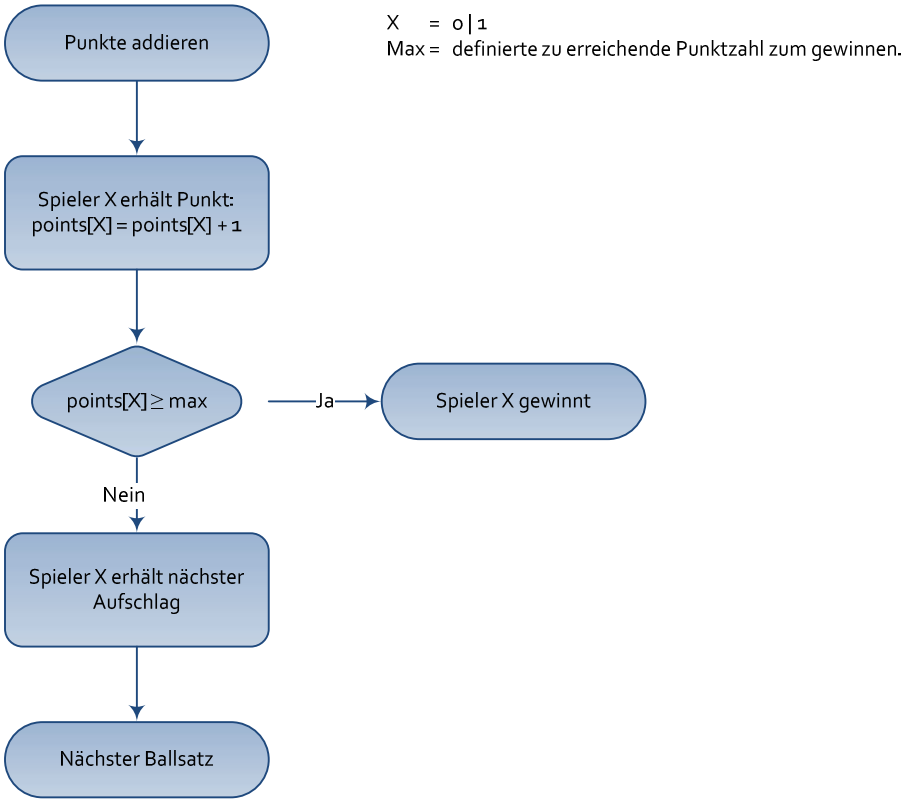


Abbildung 71: Test-Case – Punktegewinn

Tests

Um das korrekte Addieren der Punkte zu testen, haben wir das Spiel einige Male gegeneinander gespielt. Grundsätzlich funktioniert dieser Case wie erwünscht. Das Problem ist hier aber wie bereits im vorherigen Testcase 8.2.3 Ball schlagen, dass die BallStateMachine zum Teil die Kollisionen nicht richtig erhält, wobei ein korrektes Aktualisieren der Punkte nicht gewährleistet werden kann.

9 Projektmanagement

Unser Projektmanagement haben wir über das Managementtool „Redmine“ abgewickelt. Der Einsatz von Redmine hat sich für uns bewährt. Wir hatten die Phasen und einzelnen Tasks gut im Griff und waren uns über den Fortschritt des Projektes jederzeit bewusst.
Die meisten Meilensteine konnten wir halten. Was jedoch die Haptic betraf, so brauchten wir mehr Zeit als geplant. Rückblickend hätten wir die ganze Haptic im Vorfeld besser Analysieren müssen. I3D als Framework ist ein wenig eingesetztes Framework. Diesen Umstand haben wir in Form von Schwierigkeiten (Fehler konzeptionell oder Programmfehler) zu spüren bekommen. Diesen Faktor hätten wir mehr Beachtung schenken sollen.

9.1 Arbeitsteilung

Grundsätzlich haben wir alle eingesetzten Konzepte zusammenerarbeitet. Auch bei der Umsetzungen haben uns die Arbeiten zwar aufgeteilt, jedoch oftmals Probleme gemeinsam gelöst. Wir können in dem Sinne keine klare Arbeitsaufteilung machen. Die folgende Liste zeigt, welche Person welche Teile hauptsächlich gemacht hat.

Aufgaben	**Daniel Pfäffli**	**Andreas Emch**
Projektmanagement		
Projektplanung	X	X
Projektleitung	X	
Software-Komponente		
Equalizer	X	
SceneSetup	X	
SceneManager	X	
Event-System	X	
Level und Menu		X
GamePlay - BallStateMachine		X
Kollisionserkennung in Bullet		X
Aktionen für die Logik		X
Haptic	X	X
Testing		
UnitTest	X	
UseCase Tests		X

Weitere Tasks		
Poster		X
Dokumentation	X	X

Tabelle 19: Aufteilung der Arbeiten

Grundsätzlich konnten wir die Arbeiten ca. ausgeglichen halten. Anbei ein Auszug aus dem Projektmanagement-Tool Redmind:

Member	2012-9	2012-10	2012-11	2012-12	2013-1	Total
Daniel Pfäffli	22	60.57	76.25	121.75	90	370.57
Andreas Emch	25	64	94.75	102.5	91.5	377.75
Total	47	124.57	171	224.25	181.5	**748.32**

Tabelle 20: Aufwand pro Person

9.2 Zeitplan

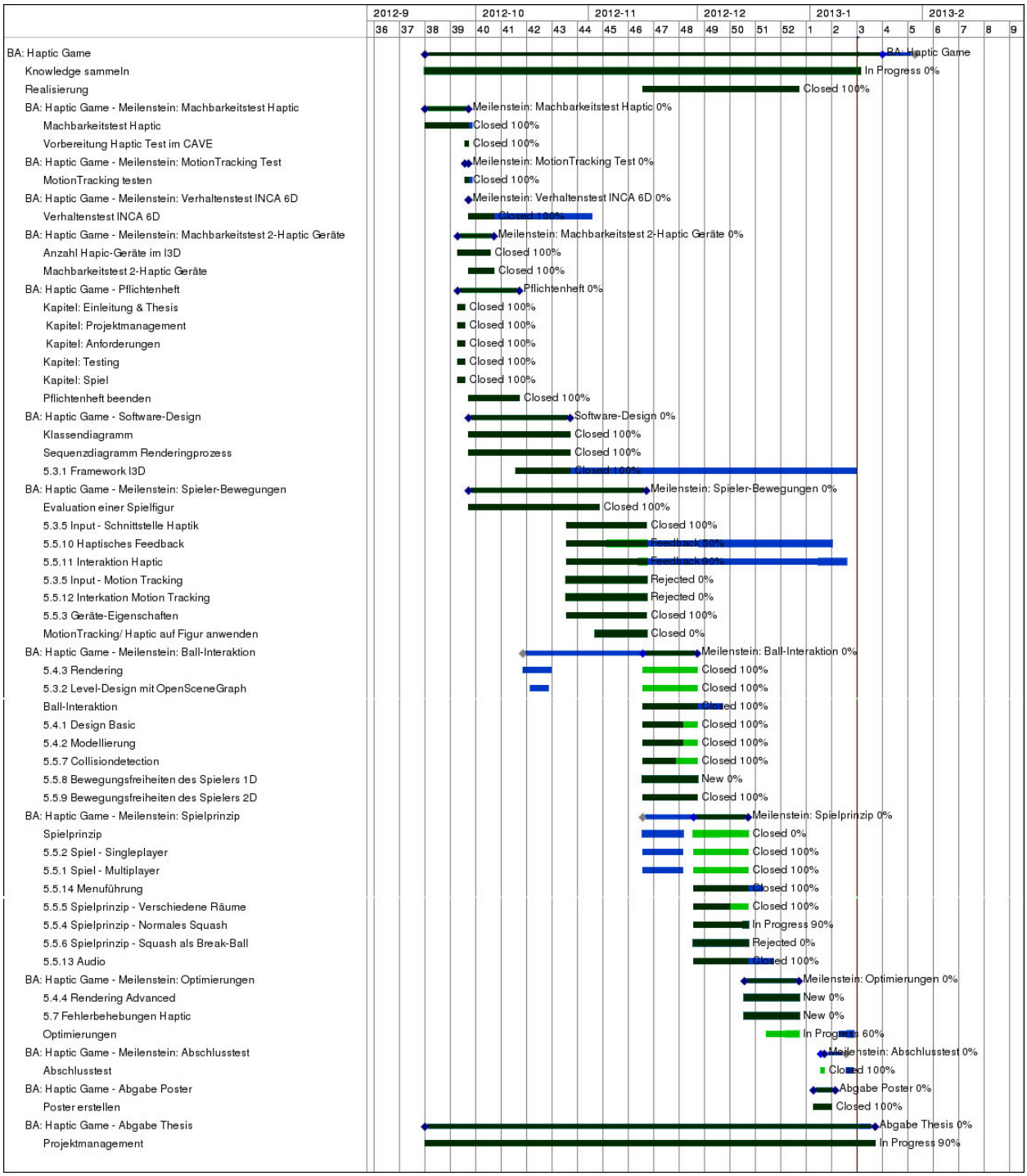


Abbildung 72: Zeitplan

Legende:
Geplant
Effektiv

Task „5.3.1 Framework I3D“ fällt deshalb aus dem Rahmen, da wir feststellen mussten, dass wir nicht alle Arbeiten auf die Tasks anwenden konnten. Arbeiten am Event-System gehörten gleichzeitig zu mehreren oder keinen Tasks. Solche Arbeiten fügten wir diesem Task hinzu.

Initialisierungsphase		**Vorgsehen**	**Effektiv**
	Projektstart	17.09.2012	17.09.2012
	Projektentscheid	21.09.2012	21.09.2012
Voranalysephase (Kaptiel gem. Pflichtenheft)			
9.5.5	MotionTracking Test	28.09.2012	29.09.2012
9.5.2	Machbarkeitstest Haptic	28.09.2012	29.09.2012
9.5.4	Verhaltenstest INCA 6D	05.10.2012	01.10.2012
9.5.3	Machbarkeitstest 2-Haptic Geräte	05.10.2012	05.10.2012
9.5.1	Pflichtenheft	12.10.2012	12.10.2012
Konzept-/ Prototyping-Phase (ggf. auch oder separate Testphase, je nach Projekt) **(Kaptiel gem. Pflichtenheft)**			
9.5.6	Software Design	26.10.2012	26.10.2012
9.5.7	Spieler-Bewegung	16.11.2012	11.01.2013
Realisierungphase (Kaptiel gem. Pflichtenheft)			
9.5.8	Ball-Interaktion	30.11.2012	08.12.2012
9.5.9	Spielprinzip	14.12.2012	22.12.2012
Abschlussphase (Kaptiel gem. Pflichtenheft)			
9.5.10	Optimierung	28.12.2012	12.01.2013
	Abschlusstest	04.01.2013	12.01.2013
9.5.11	Endprodukt	**18.01.2013**	**18.01.2013**

Tabelle 21: Meilensteine

Die grössten Diskrepanzen haben wir im Bereich „Spieler-Bewegung“. Dieser Meilenstein enthielt die Haptic. Wie dem Kapitel "5.6 PotentialFieldForceAlgo und ProxyPointForceAlgo" zu entnehmen ist, konnten wir das Problem mit dem Mesh und dem PointForce-Algorithmus nicht lösen. Deshalb blieben diese Tasks offen bis zum Ende.
Der Meilenstein Optimierung haben wir ebenfalls zu Gunsten der Haptic zurückgestellt. Dies ist sehr bedauerlich, da hierdurch ebenfalls der Spielspass leicht erhöht hätte werden können.

10 Zusammenfassung

Das Spiel SquashI3D erweitert das I3D-Framework, um die Möglichkeit beliebig viele haptische Geräte verwalten und Events (bisher waren nur statische Werte möglich) versenden zu können. Die Applikation besteht aus verschiedenen Szenen, die sich durch Spielräume unterscheiden, sowie verschiedene Menu-Szenen, welche zur Laufzeit ausgetauscht werden. Die Szenen sind dynamisch als XML-Format definiert.
Wir haben die Anforderungen Priorität 1 zu 100% erfüllt. Ausnahme bildet das haptische Feedback, welches wegen möglicher Beschädigung der Geräte abgeschaltet wurde. Die problematische Berechnung der haptischen Kräfte war nicht als Teil der Arbeit geplant. Wir haben viele Priorität 2 und ebenfalls wenige Priorität 3 Ziele umgesetzt.
Insgesamt ist es eine gelungene Arbeit und durchaus als Beta-Version eines Computerspiels vertretbar.

10.1 Erfüllungsgrad der Anforderungen

Folgend die gesetzten Anforderungen und ihren Erfüllungsgrad in Prozent.

Ziel (Kapitel gem. Pflichtenheft)		**Prio 1 Muss**	**Prio 2 Soll**	**Prio 3 Soll**
Technical Requirements				
6.3.1	Framework I3D	100%		
6.3.2	Level-Design mit OpenSceneGraph	100%		
	Varianten Input (VI)			
6.3.5	Input – Motion Tracking			0%
6.3.4	Input – Schnittstelle Haptik	100%		
	Varianten Spiel (VS)			
6.3.6	Spiel – 2-Spieler mit 2 Eingabegeräten		100%	
6.3.3	Spiel – Client-Server Architektur			0%
Visual Design				
6.4.1	Design Basic	100%		
6.4.2	Modellierung	100%		
6.4.3	Rendering	100%		
6.4.4	Rendering Advanced		60%	
Functional Requirements				
6.5.7	Collisiondetection	100%		

6.5.14	Menuführung		100%	
6.5.8	Bewegungsfreiheiten des Spielers 1D	(100%)		
6.5.9	Bewegungsfreiheiten des Spielers 2D			(100%)
6.5.13	Audio		100%	
	Varianten Input (VI)			
6.5.10	Input – Haptisches Feedback	70%		
6.5.11.1	Input – Interaktion Haptic Phantom	100%		
6.5.11.2	Input – Interaktion Haptic Inca 6D		0%	
6.5.3.1	Input – Geräte Eigenschaften Phantom	90%		
6.5.3.2	Input – Geräte-Eigenschaften Inca 6D		0%	
6.5.12	Input – Interaktion MotionTracking			0%
	Varianten Spiel (VS)			
6.5.2	Spiel – Singleplayer	100%		
6.5.1	Spiel – Multiplayer		100%	
	Varianten Spielprinzip (VSP)			
6.5.5	Spielprinzip – Verschiedene Räume	100%		
6.5.4	Spielprinzip – Normales Squash		100%	
6.5.6	Spielprinzip – Squash als Break-Ball			0%
Fehlerbehebungen				
6.7	Fehlerbehebungen		0%	
Total		96.9%	62.2%	20.0%

Tabelle 22: Zielerfüllung

Insgesamt ist die Zielerfüllung sehr zufriedenstellend. Einzig beim haptischen Feedback, was als Priorität 1 gesetzt war, müssen wir Abstriche machen. In unserer Projekt-Kalkulation haben wir nicht damit gerechnet, dass der I3D-eigene Haptic-Algorithmus angepasst werden muss (siehe 5.6 PotentialFieldForceAlgo und ProxyPointForceAlgo).

Ebenfalls haben wir die Ziele, welche eine Spielfigur betreffen, nicht erfüllt. Das Einsetzen einer Spielfigur hätte einerseits einen grossen Mehraufwand für ein ansehnliches Resultat bedeutet, anderseits hätte der Spieler neben dem haptischen Gerät ebenfalls die Spielerfigur steuern müssen. Dies erschien uns zu komplex. Siehe 5.7 Einsatz einer Spielerfigur.

Durch das bewusst allgemein gehaltene Software-Design, können mit wenig Aufwand die allgemeinen Features (EventSystem, Haptic, Menu) von SquashI3D ins Framework I3D übernommen werden.

11 Fazit

Das Projekt SquashI3D gehört zu einem der bisher herausforderndster Projekte, die ich umgesetzt habe. Zu meinen sonstigen Projekten zählen CRM-Projekte, wo die Herausforderung meistens im Kunden und nicht im eigentlichen Projekt liegt.
Mit SquashI3D konnte ich erlerntes aus meiner Spezialisierungsrichtung CPVR, sowie anderer an der Berner Fachhochschule besuchter Module verbinden. Besonders hervorheben möchte ich die Module Paralleles Rechnen und Spieltheorie, welche uns bei der Entwicklung unserer Konzepte geholfen haben.
Es war interessant mit einer Umgebung wie dem CAVE zu arbeiten. Für einen solchen Raum ein Projekt umsetzen zu können, hat mich immer wieder motiviert. An dieser Stelle muss auch gesagt sein, dass die Arbeit mit dem CAVE alles andere als einfach ist. Unerklärliche Abstürze oder Verhalten gehörten zur Tagesordnung im CAVE.
Die Arbeit mit einem haptischen Gerät ist etwas, was ich gerne fortführen möchte. Es fasziniert mich, dass eine 3D-Welt auch haptische wahrgenommen werden kann. Leider sind diese Geräte noch immer sehr unausgereift und für den täglichen Gebrauch viel zu teuer.
Im Speziellen möchte ich mich bei Claudia Weber bedanken, die sich Zeit genommen hat, uns in die Haptic-Implementation des I3D-Frameworks einzuführen.

Daniel Pfäffli

Die Programmierung in C/C++ hat mich schon vor einigen Jahren fasziniert. In dieser Arbeit konnte ich endlich ein grösseres Projekt damit umsetzten und somit meine Fähigkeiten darin durch den intensiven Gebrauch sehr Vertiefen.
Bereits in der Vertiefung CPVR hatte mich die Programmierung im 3D-Raum angesprochen. Mir gefallen das räumliche Denken und die Umsetzung von 3D-Anwendungen. Gerne hätte ich mir mehr Zeit genommen für die Modellierung von verschiedenen Räumen oder Spielmodellen. Aufgrund der Prioritäten fand dies keine Zeit in unserer Umsetzung.
Die Haptik habe ich ebenfalls erst in dem Modul für CPVR kennen gelernt. Schon im Unterricht hatte ich mir Gedanken darüber gemacht, wie diese wohl am besten umzusetzen ist und ob das Phantom Omni überhaupt tauglich ist für den Heimgebrauch. Ich persönlich finde es sehr schade, dass wir das Haptik-Feedback nicht so weit umsetzten konnten, wie es geplant war. Dank dieser Arbeit konnte ich mich tief in die Programmierung von haptischen Geräten, speziell dem Phantom Omni, befassen.
Die Umsetzung im CAVE fand ich interessant, weil wir unser Spiel noch zusätzlich auf die Parallelität umsetzten mussten. Dies war zwar ein grosser Zeitaufwand, doch wenn ich das Resultat im CAVE sehe, dann hat sich dieser Aufwand wirklich gelohnt.
Insgesamt bin ich glücklich über die erreichten Ziele, deren Umsetzung und das schlussendliche Resultat. Einzig das haptische Feedback wäre noch einen Extra-Punkt, um das Spiel komplett abzurunden. Ich habe viel in den verschiedenen Gebieten unserer Arbeit gelernt und hoffe, dass ich das Gelernte immer wieder brauchen kann.

Andreas Emch

Quellverzeichnis

Abbildung 4: Darstellung CAVE
https://www.cpvrlab.ti.bfh.ch/wiki/_media/huce:cpvrlab:cave:cave_complete_3d_1280px.jpg

Abbildung 6: SensAble Phantom Omni
http://www.sensable.com/documents/images/LargePHANTOMOmniImage.jpg

Abbildung 41: Input und Output der Haptic Einzelspieler
http://www.sensable.com/documents/images/LargePHANTOMOmniImage.jpg

Abbildung 42: Input und Output der Haptic Mehrspieler
http://www.sensable.com/documents/images/LargePHANTOMOmniImage.jpg

Abbildung 43: Input und Output der Haptic Mehrspieler über Netzwerk
http://www.sensable.com/documents/images/LargePHANTOMOmniImage.jpg

Glossar

Node/ Nodes
: Sofern nicht anders erklärt, bezeichnet dieser Begriff einen Computer im CAVE-System. Der Begriff kommt aus dem Equalizer-Framework, wo pro Computer eine sognannte Node erstellt wird.

GPU
: Graphic Processor Unit, bezeichnet den Teil des Computers, welcher die Bildschirm-Ausgabe berechnet.

OSG
: OpenSceneGraph, bezeichnet eine Art, wie in OpenGL eine virtuelle Welt beschrieben werden kann.

CAVE
: Computer Automated Virtual Environment, bezeichnet ein Raum bestehend aus vier Leinwänden, welcher das Begehen einer 3D-Welt erlaubt.

BFH
: Berner Fachhochschule

DoF
: Freiheitsgrade (engl. Degrees of Freedom) wird bei haptischen Geräten verwendet, um anzugeben, wie beweglich das Gerät ist.

Rendern
: Bezeichnet den Prozess, der aus Rohdaten ein Bild erzeugt.

Scenegraph
: Stellt einen Graphen dar. Der Graph bildet eine virtuelle Szene ab, wobei die Knoten, vereinfacht gesagt, jeweils Objekte und ihre Positionen darstellen.

Literaturverzeichnis

Beginner's Guide
: Rui, Wang und Qian, Xuelei. OpenSceneGraph 3.0 - Beginner's Guide. Birmingham - Mumbai : Packt Publishing Ltd., 2010. ISBN 978-1-849512-82-4.

Cookbook
: Rui, Wang und Qian, Xuelei. OpenSceneGraph 3 - Cookbook. Birmingham - Mumbai : Packt Publishing Ltd., 2012. ISBN 978-1-84951-688-4.

OpenSceneGraph
: www.openscenegraph.com. 2012/13

Physics Simulation Wiki
: http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page. 2012/13

Bonzai Software - Water-Fire-Sky
: http://www.bonzaisoftware.com/wfs.html. 01.2013

Sensable
: http://www.sensable.com/ 2012/13

huce:cpvrlab:start
: Wiki :: Institut für Human Centered Engineering :: BFH-TI. https://www.cpvrlab.ti.bfh.ch/wiki/huce:cpvrlab:start. 2012/13

Equalizer
: http://www.equalizergraphics.com, 2012/13

Bachelor Arbeit CAVE Haptic
: CAVE Haptic, V-Touch Library, Del Piero Michel, Weber Claudia, 2010/11

Anhang

Weitere Dokumentationen zum Projekt SquashI3D finden sich auf der DVD (siehe 1.4 Inhalte der DVD). Folgende Dokumentationen stehen zur Verfügung:

- Arbeitsjournal
- Sitzungsprotokolle
- Code-Dokumentation

Weitere Dateien:

- Projekt Sources
- Kompilierte Version von SquashI3D

Erklärung der Diplomandinnen und Diplomanden

Rechte an der Bachelor Thesis

Ich bin damit einverstanden, dass die BFH-TI alle Güter inklusive der Immaterialgüter der durch die BFH-TI geleiteten Arbeiten grundsätzlich nutzen darf. Besondere Abmachungen zwischen Dozenten und beteiligten Unternehmungen und Institutionen bleiben vorbehalten.

Selbständige Arbeit

Ich bestätige mit meiner Unterschrift, dass ich meine Bachelor Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.), die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt.

Datum	17.01.2013
Name Vorname	Andreas Emch
Unterschrift	.....
Name Vorname	Daniel Pfäffli
Unterschrift	.....