ABYSS

Ein frei erfundenes 2D-Computerspiel

Projektarbeit im zweijährigen Berufskolleg zum
Kommunikations- und Informationstechnischen Assistent

Inhaltsverzeichnis

Persönliche Erklärung

1. Ich versichere, dass ich die vorliegende Arbeit mit dem Thema: ABYSS selbständig verfasst und angefertigt habe. Ich habe nur die angegebenen Hilfsmittel benutzt.

   Die Stellen, die anderen Werken dem Wortlaut oder dem Sinne nach entnommen sind, habe ich in jedem einzelnen Fall durch Angaben der Quelle als Entlehnung kenntlich gemacht.
2. Diese Arbeit wird nach Abschluss an der Staatl. Feintechnikschule VS für den Tag der offenen Tür am 26.07.2009 zur Verfügung gestellt.

   Datum, Ort: 16.06.2009, Villingen-Schwenningen

   Unterschrift:

Einleitung

Die Projektarbeit an der Feintechnikschule ist zur Förderung der Schüler in verschiedenen Bereichen gedacht. Diese sollen lernen eine Idee umzusetzen. Auf diesem Weg ist es Ziel zu lernen, zu planen, zu gestalten, Probleme zu lösen, Informationen zu beschaffen und letzten Endes eine Dokumentation anzufertigen sowie das Projekt zu präsentieren. Auf diese Art lernt der Schüler verantwortungsbewusst, kreativ, und selbständig zu arbeiten.

Beschreibung der Projektarbeit

Im Folgenden wird die Projektarbeit zusammenfassend beschrieben, um einen Überblick zu ermöglichen.

Problemstellung

Ziel dieser Projektarbeit ist das Programmieren eines frei erfundenen Computerspiels in C# (gesprochen: C-Sharp). Als Szenario wird hierfür der Weltraum verwendet. Der Spieler soll mit einem Raumschiff auf einer definierten Spielfläche feindliche Raumschiffe zerstören können. Für diesen Zweck soll das Spielerraumschiff verschiedene Waffen besitzen können. Außerdem soll es über Powerups[1] möglich sein, das Raumschiff und dessen Waffen zu verbessern.

Projektrahmen / Projektumfang

Das Projekt umfasst die vollständige Programmierung eines Spielprinzips. Verwendete Medien wie z.B. Bilder werden daher nicht selber gestaltet, da dies den Rahmen der Projektarbeit überschreiten würde. Auch wird eine fremde Grafik-Engine[2] benutzt um die Aufgabe in dem definierten Zeitrahmen zu bewerkstelligen.
Einzelne Ausnahmen sind vorbehalten. So sind z.B. die Bilder der Powerups selbst erstellt.
Des weiteren waren für das Projekt optionale Erweiterungen geplant. Die Erweiterung, das Spiel zu vertonen (mit Musik und Soundeffekten zu hinterlegen) scheiterte an mangelnder Zeit, die Erweiterung mehr als fünf Powerups zu gestalten gelang jedoch.

1) „Powerup ist ein Begriff aus der Welt der Computerspiele, sowohl im PC-Bereich als auch auf Konsolen und Spielautomaten. Er bezeichnet ein Objekt, das die Spielfigur meist durch einfache Berührung aufnehmen kann und das sie auf irgendeine Weise für einen begrenzten Zeitraum mit besonderen Fähigkeiten ausstattet – sie läuft schneller, springt höher oder schießt genauer.“
(Quelle: http://de.wikipedia.org/wiki/Powerup)

2) „Eine Grafik-Engine (wörtlich „Grafik-Maschine“, freier etwa: „Grafiktriebwerk“ oder „Grafikmodul“) ist ein mitunter eigenständiger Teil eines Computerprogramms oder einer Computer-Hardware, welcher für dessen Darstellung von Computergrafik zuständig ist, meist möglichst realitätsgetreuer 3D-Computergrafik, wie Gegenstände, Umwelt und Personen (Stichwort: Virtuelle Realität). Im Zusammenhang mit 3D-Computergrafik bezeichnet man die Grafik-Engine daher dann auch als 3D-Engine. Konkret handelt es sich dabei um einen integrierten oder extern gelagerten Programmcode, der parallel zum eigentlichen Spiel (siehe Game-Engine) für die Berechnung der Grafikschnittstelle zuständig ist.“
(Quelle: http://de.wikipedia.org/wiki/Grafik-Engine)

Projektname

„Abyss“, aus dem Englischem, wird zwar gerne mit mit „Abgrund“, „Kluft“ oder „Schlund“ ins Deutsche übersetzt, es heißt aber auch „Unendlichkeit“. Da der Name kurz, „wohlklingend“, zum Weltraumszenario passend und sogar etwas poetisch ist, wird er als Name des Spiels und als Titel der Projektarbeit verwendet.

Aufbau des Spiels

Um im Späteren das genaue Zusammenspiel der Klassen und Objekte zu erläutern, muss zunächst erstmal der Aufbau des Spiels genauer betrachtet werden.

Grafik-Engine

Wie zuvor schon erwähnt ist die Programmierung einer Grafik-Engine nicht Ziel dieser Projektarbeit. Es kommt daher die schon vorhandene, freie und Open Source[3] Engine „Irrlicht“ zum Einsatz.

Allgemeines

Die Irrlicht Engine ist ein nicht kommerzielles Projekt, von einem internationalen Entwickler-Team. Seit 2002 wird sie bis heute ständig weiterentwickelt, was ihr eine große und hilfsbereite Community[4] einbrachte. Des weiteren ist sie plattformunabhängig und läuft daher auf Linux, Mac OS, Sun Solaris und diversen Windows-Versionen.
Da die Engine in C++ entwickelt wird, ist für ihre Verwendung in einer .NET Sprache wie C# ein Wrapper[5] zwingend notwendig. Dieser Wrapper wurde aber nur bis zur Irrlicht Version 1.3.1 entwickelt. Daher wird das Spiel auf diese, etwas veraltete, Version aufgebaut. (Aktelle Version: Irrlicht 1.5.0, Stand 26.05.09)

3) „Open Source [ˈoʊpən sɔːɹs] (engl.) bzw. quelloffen ist eine Palette von Lizenzen für Software, deren Quelltext öffentlich zugänglich ist und durch die Lizenz Weiterentwicklungen fördert.“ (Quelle: http://de.wikipedia.org/wiki/Open_Source)
4) „Eine Online-Community (Netzgemeinschaft) ist eine Sonderform der Gemeinschaft; hier von Menschen, die einander via Internet begegnen und sich dort austauschen.“ (Quelle: http://de.wikipedia.org/wiki/Online-Community)
5) „Als Wrapper bezeichnet man in der EDV allgemein ein Programm, das als Schnittstelle zwischen dem aufrufenden und dem umschlossenen (engl. wrapped) Programmcode agiert. Dies kann aus Kompatibilitätsgründen eingesetzt werden, wenn beispielsweise der umschlossene Code eine andere Programmiersprache verwendet [...].“ (Quelle: http://de.wikipedia.org/wiki/Wrapper_(Software))

Aufgaben der Engine im Projekt

Die Engine ist dafür da, Objekte wie gewollt auf den Bildschirm darzustellen. Es handelt sich in diesem Projekt hauptsächlich um 2D-Grafiken. Im Menü finden jedoch auch Mesh-Objekte[6] ihren Einsatz. Diese sind dann dreidimensional.
Eine weitere Aufgabe der Engine sind einfache Animationen. So kann sie Objekte von A nach B bewegen, oder Partikeleffekte erzeugen die ein Feuer nachahmen.
Als letztes wird noch die Kollisionserkennung von Irrlicht in Anspruch genommen. Eine Kollisionserkennung ist kein zwingender Bestandteil einer Grafik-Engine, aber da diese ausreichend für das Projekt ist, wurde sie auch angewendet.

Medien

Medien wie Bilder, Musik und Soundeffekte werden für das Spiel benötigt. Wie im Kapitel „Projektrahmen / Projektumfang“ beschrieben, werden diese nicht selbst kreiert, sondern aus dem Internet bezogen. Der Einsatz von Musik und Soundeffekten ist im Quelltext vorgesehen und berücksichtigt, wurde aber mangels Zeit nicht mehr fertig umgesetzt.

Logo- und Menüdesign

Das Logo des Spiels wurde auf der Internetseite „http://cooltext.com/“ erstellt und mit GIMP überarbeitet. Das futuristische Design findet sich auch im Rest des Menüs wieder.
Als Menühintergrund wir eine 3D-Szene gezeigt in der sieben Planeten, ein Zwergplanet und ein Mond aus unserem Sonnensystem dargestellt sind (Mars, Merkur, Venus, Erde, Mond, Saturn, Jupiter, Neptun und Pluto). Ihre Texturen[7] entsprechen dem Original. Eine bläuliche Sonne umkreist die Himmelskörper und wirft realistisch Licht auf sie. Die Kamera fliegt dabei durch den Raum von Objekt zu Objekt.
(Siehe Abbildung 1)

Spieldesign

Im Spiel ist das Design anders als im Menü. Hier sehen die Raumschiffe zwar modern aus, aber das Head-Up-Display[8] (kurz: HUD) ist alt gehalten und die Schrift erinnert an 7-Segment Anzeigen. Die Raumschiffe sind überarbeitete Bilder von 3D-Modellen, die von oben aufgenommen wurden. Sie stammen von einem 3D-Modelle Vertreiber aus dem Internet (http://www.turbosquid.com/).
Der Weltraum besteht aus real existierenden Galaxien, Nebel, Planeten und Asteroiden. Diese Bilder gibt es zur freien Verfügung auf der Webseite des Hubble-Weltraumteleskop. (http://hubblesite.org/).
(Siehe Abbildung 2)

---

6) „Mesh (engl.) in der Computergraphik bezeichnend für ein Polygonnetz zur Modellierung von 3D Objekten [...].“
(Quelle: http://de.wikipedia.org/wiki/Mesh)
7) „Im Bereich der Computergrafik verwendet man Texturen als "Überzug" für 3D-Modelle um deren Detailgrad zu erhöhen ohne den Detailgrad der Geometrie zu erhöhe.“
(Quelle: http://de.wikipedia.org/wiki/Textur_(Computergrafik))
8) „Das Head-up-Display (HUD; sinngemäß: Anzeigefeld in Blickrichtung) ist ein Anzeigesystem, bei dem die für den Nutzer (Piloten, neuerdings auch Autofahrer etc.) wichtigen Informationen in sein Sichtfeld projiziert werden.“
(Quelle: http://de.wikipedia.org/wiki/Head-Up-Display)

Abbildung 1: Hauptmenü

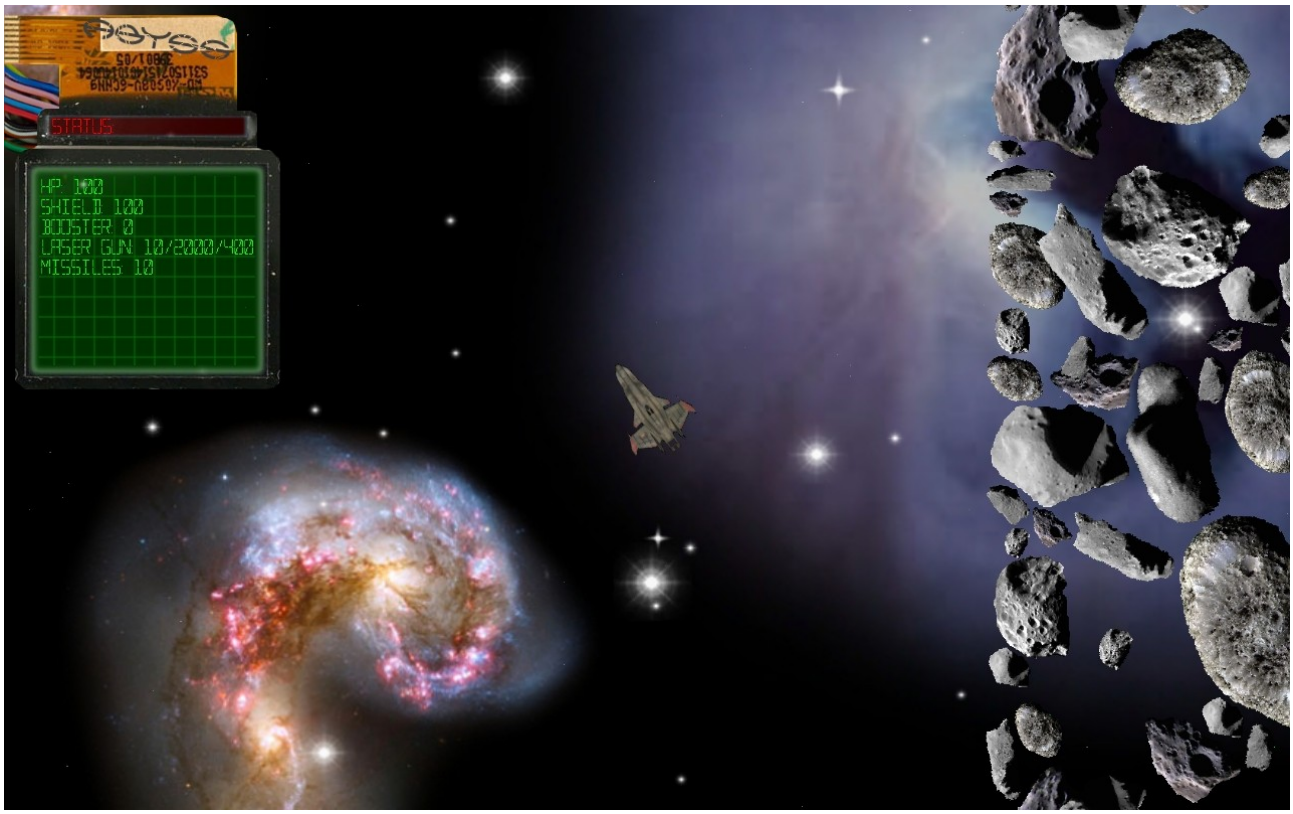

Abbildung 2: Weltraum (Links: HUD, Mitte: Spielerraumschiff, Rechts: Spielfeldgrenze)

Bildbearbeitung

Da die Bilder vierkantig sind, die Objekte aber, die sie darstellen (z.B. ein Raumschiff), ganz anderes geformt sind, mussten alle Bilder (mit Ausnahme der Himmelskörper-Texturen) manuell nach bearbeitet werden. Das heißt, alles was von dem Bild im Spiel nicht mehr sichtbar sein sollte, wurde als transparent gekennzeichnet.

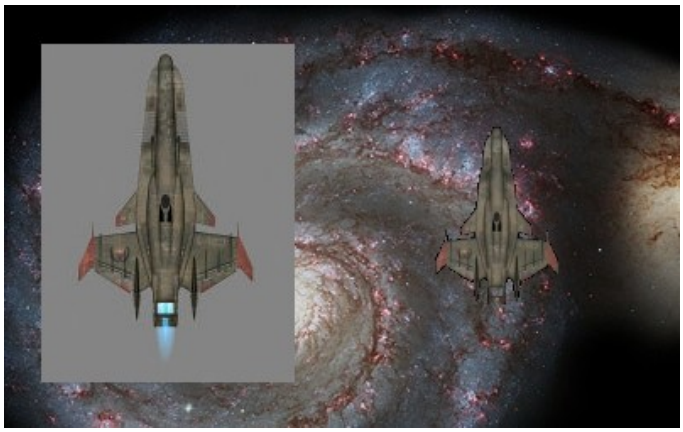

Abbildung 3: Links: Originalgrafik, Rechts: Bearbeitete Grafik mit Transparenz

Spielprinzip

Das Spielprinzip ist recht simpel gehalten, jedoch ausbaufähig. Klassisches Missionsziel ist das Zerstören aller feindlichen Raumschiffe im Weltraum. Wenn dies dem Spieler gelingt, so hat er gewonnen, oder kommt in ein weiteres Level. Gelingt es ihm nicht, weil er zerstört wurde oder gegen einen Asteroiden geflogen ist, so verliert er das Spiel.

Spielsteuerung

Das Flugverhalten der Raumschiffe ist etwas Besonderes in diesem Spiel. Im Gegensatz zu vielen anderen 2D-Raumschiffspielen, ist es hier möglich in alle vier Himmelsrichtungen zu fliegen. Das Schiff verhält sich dabei wie wenn es auf Eis „schlittern“ würde. Das heißt wenn man zuerst nach Norden beschleunigt und dann das Raumschiff nach Osten dreht und wieder beschleunigt, so fliegt dieses Richtung Nord-Osten, bis es irgendwann nur noch nach Osten fliegt.

Taste	**Wirkung**
↑ (Pfeiltaste nach oben)	Beschleunigen
↓ (Pfeiltaste nach unten)	Bremsen
← (Pfeiltaste nach links)	Nach links drehen
→ (Pfeiltaste nach rechts)	Nach rechts drehen
TAB (Tabulator)	HUD umschalten (Status, Mission, Position)
F	FPS (Frames Per Second) anzeigen
Esc	Hauptmenü aufrufen

Tabelle 1: Tastenbelegung des Spiels

Wichtige Klassen (und Objekte)

An dieser Stelle werden einige Klassen erläutert. Da es für das Verständnis überflüssig wäre auf jede Klasse einzugehen, werden nur ein paar wichtige erklärt. Wer jedoch, in zusammenfassenden Worten, für jede Klasse eine Beschreibung möchte, findet diese (in Englisch) im Quellcode.

Globale Objekte

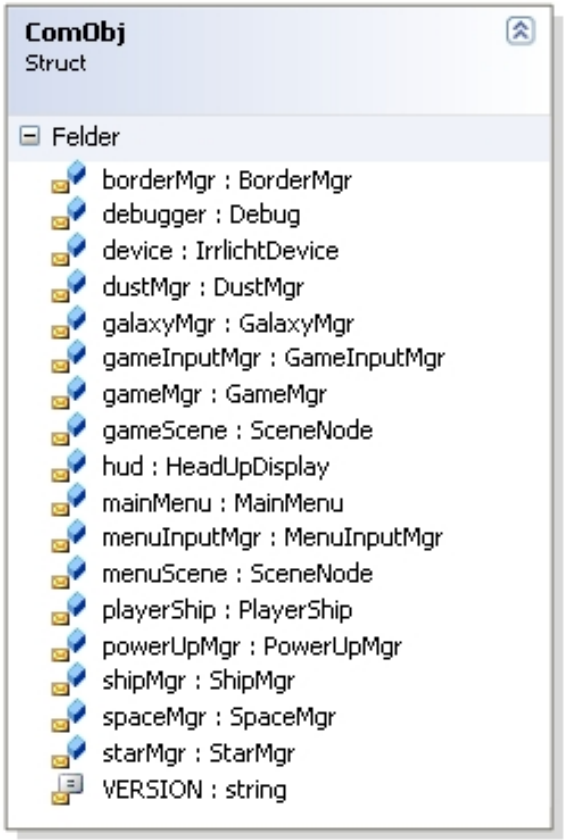

Abbildung 4: Globale Objekte des Spiels

Die Abbildung 4 zeigt alle globalen Objekte des Spiels. Bis auf die „VERSION“ wird hier auf alle Objekte, der Reihenfolge nach eingegangen.

Der Border Manager (BorderMgr)

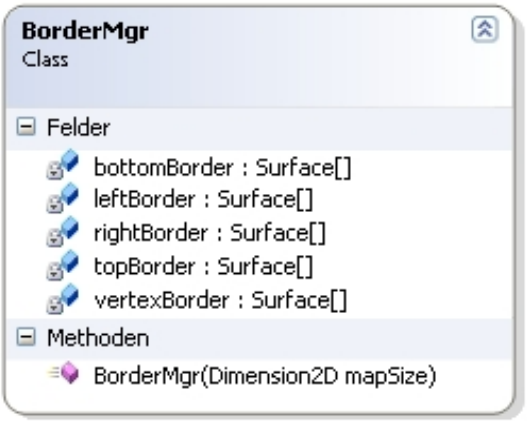

Abbildung 5: Klasse BorderMgr

Der Border Manager ist für den Aufbau der Spielfeldgrenzen zuständig. Die Grenze selber besteht aus der Oberseite (topBorder), Unterseite (bottomBorder), linken Seite (leftBorder) und der rechten Seite (rightBorder). In allen der vier Ecken wird eine Extragrafik gesetzt (vertxBorder). Die vier Seiten bestehen jeweils aus einer Wiederholung der selben Grafik. Diese wird einfach immer um 90° pro Seite gedreht.
Die Klasse wird zu Beginn eines Levels initialisiert und benötigt nur die Spielfeldgröße. Eine Größe von 2x3 entspricht der Größe von zwei mal Grafik 2 auf drei mal Grafik 2

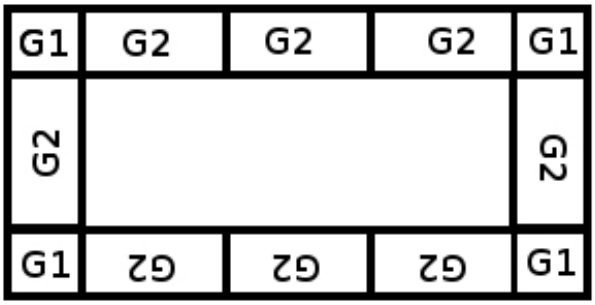

Abbildung 6: Spielfeldrahmen mit Größe 3x1 (G1 / G2 = Grafik 1 / 2)

Der Debugger (Debug)

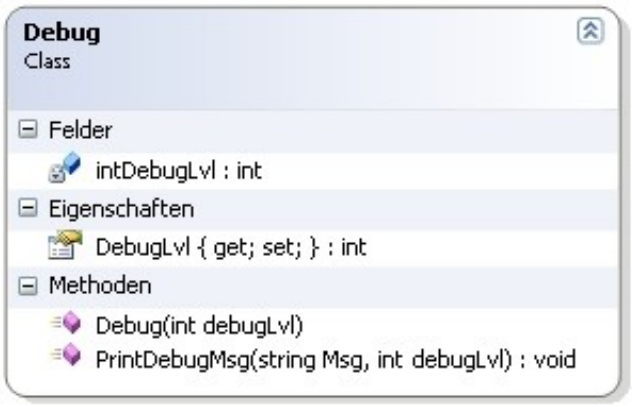

Abbildung 7: Klasse Debug

Die Debug Klasse wird mit dem Start des Spiels geladen. Sie verfügt über ein Level das bestimmt, ob eine Debug-Meldung ausgegeben werden soll oder nicht. Sie ist sozusagen ein Filter, der prüft ob die Informationsausgabe gewollt ist oder nicht. Das Debug-Level kann beim Start des Spiels mit dem Parameter „-d X“ übergeben werden.
(z.B. abyss.exe -d 2)

Debug-Level (X)	Wirkung
0	Nur Engine bedingte Informationen werden ausgegeben
1	Wichtigste Debug-Meldungen werden ausgegeben
2	Debug-Meldungen zum Spielverlauf werden ausgegeben
3	Sehr viele Debug-Meldungen über alles was im Spiel passiert

Tabelle 2: Wirkung der verschiedenen Debug-Level

Das Irrlicht Device (IrrlichtDevice)

Das IrrlichtDevice ist sozusagen Kernschnittstelle zu Grafik-Engine. Ob es um das Laden von Texturen, das Erstellen einzelner Mesh-Objekte, die Darstellung von Schrift oder das Rendern der ganzen Szene geht, so ist immer wieder auf das IrrlichtDevice zurück zu greifen.

Der Dust Manager (DustMgr)

Abbildung 8: Klasse DustMgr

Diese Klasse ist für die „Staubpartikel“ im Weltraum verantwortlich. Sie wird beim Level-Start initialisiert. Hier wird festgelegt, wie viele Staubkörner es um das Spielerraumschiff gibt (mit einem gewissen Puffer über den Bildschirmrand hinaus). Wenn das Raumschiff nun fliegt, so berechnet dieser Manager die Länge und die Drehung der einzelnen Staubpartikel. Er kontrolliert außerdem das Staubkörner, die aus dem Bereich fallen, auf der anderen Seite wieder zufällig positioniert werden.

Der Galaxy Manager (GalaxyMgr)

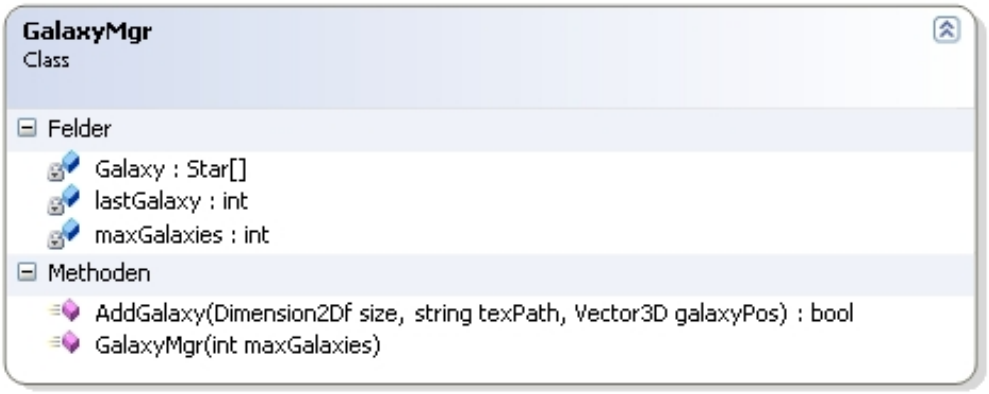

Abbildung 9: Klasse GalaxyMgr

Beim Start des Levels wird dem Galaxy Manager „gesagt“ wie viel Galaxien er zu verwalten hat. Anschließend bekommt er für jede einzelne Galaxie „mitgeteilt“ wie groß sie ist, wo sie im Weltraum platziert werden soll und den Pfad zu der Grafik.

Der Game / Menu Input Manager (GameInputMgr / MenuInputMgr)

Der Game und der Menu Input Manager ähneln sich von der Funktion her sehr und werden deshalb im Folgendem zusammengefasst.

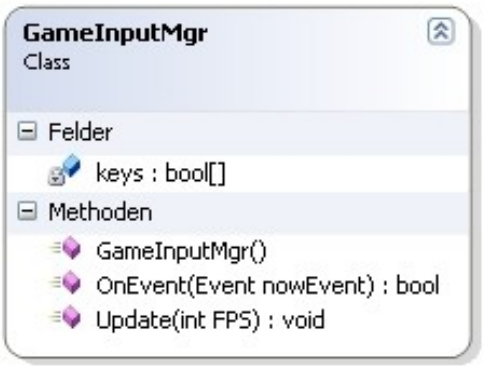

Abbildung 10: Klasse GameInputMgr

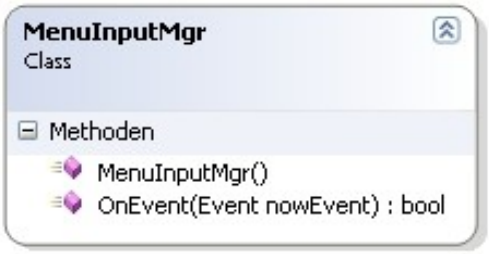

Abbildung 11: Klasse MenuInputMgr

Diese Klassen unterscheiden sich nicht in der Anwendung, lediglich in Struktur und dem Zeitpunkt ihrer Anwendung. Beide sind dafür verantwortlich, die vom Spieler gedrückten Tasten entgegen zunehmen und zu verarbeiten. Der Menu Input Manager tut dies während der Spieler im Menü ist und einen Eintrag wählen kann. Der Game Input Manager reagiert auf Tasten während des eigentlichen Spiels.
(Zur Tastenbelegung siehe Kapitel „Steuerung / Gameplay)

Der Game Manager (GameMgr)

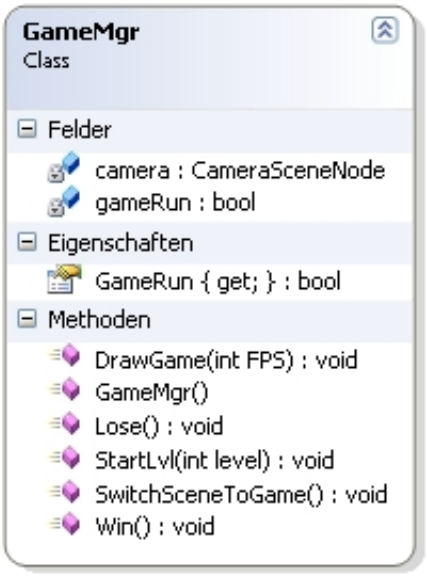

Abbildung 12: Klasse GameMgr

Wie dem Namen der Klasse schon zu entnehmen ist, ist der Game Manager für die Abwicklung des eigentlichen Spiels zuständig. Wenn der Spieler beispielsweise ein neues Level „betritt“ oder stirbt (z.B. da er mit einem Asteroiden kollidiert ist), so regelt diese Klasse was zu tun ist. Die wohl komplexeste Methode von ihr ist die StartLvl(int level). Sie initialisiert alle Komponenten für das Spiel. Beispiele hierfür sind Space Manager, der Ship Manager, das Spielerraumschiff, das HUD und einige weitere.

Die gameScene und die menuScene (SceneNode)

Die beiden Objekte bestehen aus der gleichen Klasse: SceneNode. Diese Klasse ist von der Grafik-Engine gegeben. Sie ist eine Basisklasse, die jedes Objekt einer 3D-Szene[9] beschreiben kann. In diesem Fall wird sie wie eine Art Schublade verwendet. So sind gameScene und menuScene jeweils eine Schublade, in der die zugehörigen Objekte der Szene „abgelegt“ werden.

Beispiel:
gameScene „enthält“ das Spielerraumschiff, da dies im eigentlichen Spiel gebraucht wird.
MenuScene „enthält“ die Himmelskörper, da diese im Menühintergrund angezeigt werden.

Vorteil davon ist, dass alle Objekte in einer Schublade, vereinfacht, verwaltet werden können. Wenn beispielsweise das ganze Menü unsichtbar gemacht werden soll, so reicht es die menuScene als unsichtbar zu markieren.

9) Eine Szene beschreibt in der 3D-Programmierung eine Konstellation aus Objekten (auch: Szenen-Knoten).
So wird z.B. das Hauptmenü dieses Spiels mit Himmelskörpern, Lichtquelle, Kamera und Text zusammenfassend als Szene bezeichnet.

Das Head-Up-Display (HeadUpDisplay)

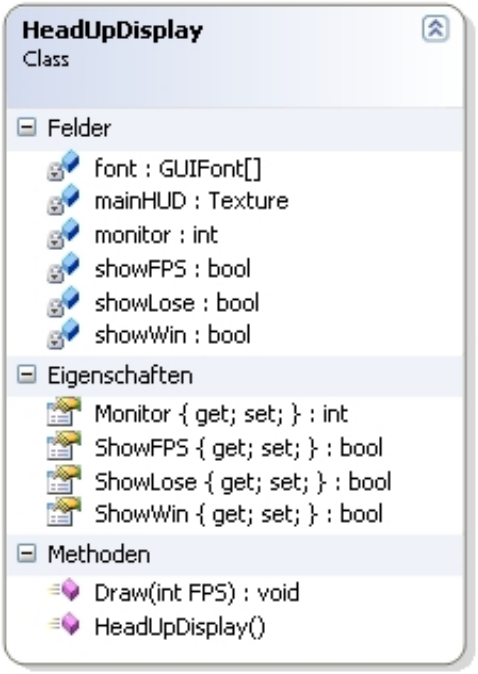

Abbildung 13: Klasse HeadUpDisplay

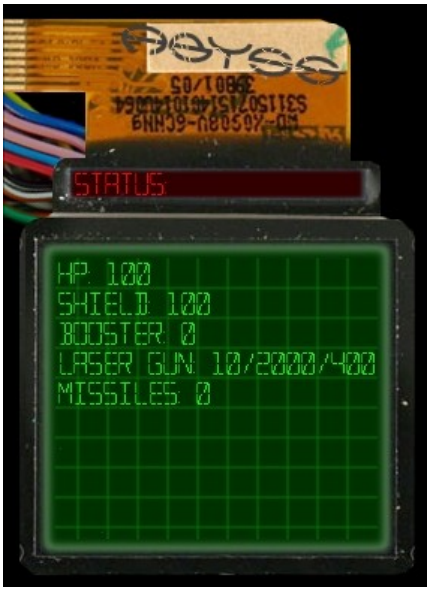

Abbildung 14: Head-Up-Display im Spiel (Status Monitor)

Das Head-Up-Display ist dazu da, Informationen auf dem Bildschirm darzustellen. Die meisten Informationen werden links oben in einem virtuellem Display / Monitor angezeigt (siehe Abbildung 14).

Eigenschaft	Wert
HP (Health Points / Lebensenergie)	Zahlenwert zwischen 0 und 100 (0 = Spieler ist tot)
Shield	Zahlenwert zwischen 0 und 100 (Schutzschildenergie)
Booster	Zahlenwert zwischen 0 und „unendlich“ (Nachbrennerenergie um schneller zu fliegen)
Laser Gun (Primäre Waffe, Laser)	X / Y / Z X = HP Schaden (X/2 = Shield Schaden) Y = Projektilgeschwindigkeit (Zeit in Millisekunden bis Projektil sein Ziel erreicht) Z = Feuerrate (Zeit in Millisekunden bis Waffe wieder einsatzbereit ist)
Missiles (Sekundäre Waffe, Raketen)	Zahlenwert zwischen 0 und unendlich entspricht der Anzahl verfügbarer Raketen

Tabelle 3: Werteerklärung für den Monitor in „Status“ Einstellung

Des weiteren kann dieser Monitor mit der Tabulatortaste umgeschaltet werden.

Einstellung	Ausgabe
Status	Ausgabe des aktuellen Zustands des Spielerraumschiffs (siehe Abbildung 14 und Tabelle 3)
Mission	Informationen über Missionsziel
Coordinates	Ausgabe der aktuellen Position des Spielerraumschiffs

Tabelle 4: Monitor Ausgabemöglichkeiten

Die HeadUpDisplay Klasse ist aber auch für die Ausgabe an Textinformationen außerhalb des virtuellen Monitors verantwortlich. Wenn die FPS[10] dargestellt, oder der „You Win“ Schriftzug eingeblendet werden soll, so regelt auch dies jene Klasse.

10) „Die Abkürzung FPS (für das englische Frames per Second) bezeichnet bei Film- und Videoaufnahmen sowie bei graphischen Computeranwendungen die Anzahl der Bilder pro Sekunde.“ (Quelle: http://de.wikipedia.org/wiki/Bildfrequenz)

Das Hauptmenü (MainMenu)

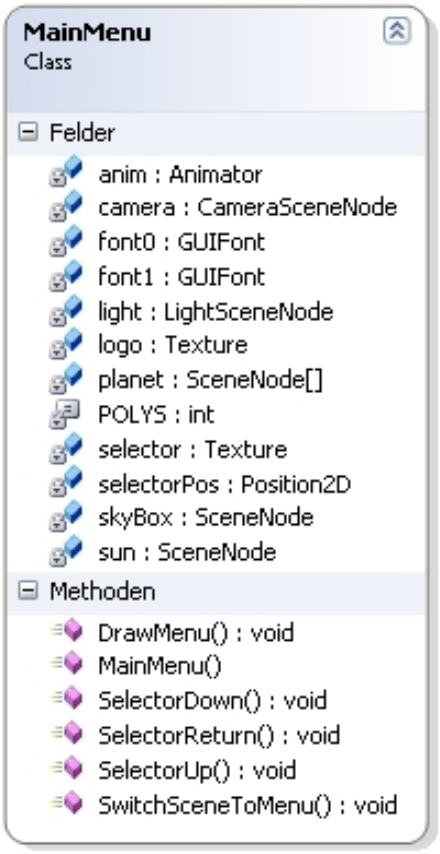

Abbildung 15: Klasse MainMenu

Diese Klasse stellt das ganze Menü dar. Sie lädt die komplette Szene, wie im Kapitel „Logo- und Menüdesign“ beschrieben, bei der Initialisierung ein.

Die meisten Funktionen für diese 3D-Szene im Menü bietet die Grafik-Engine schon. So ist beispielsweise die Sonne eine 2D-Grafik, die sich immer der Kamera entgegen dreht. An der selben Stelle der Sonnen-Grafik ist auch eine Lichtquelle, die für das dynamische Licht sorgt. Die Planeten sind von der Engine erstellte 3D-Kugeln, die mit der entsprechenden Textur überzogen werden. Als Rahmen der ganzen Szene dient die sogenannte SkyBox. Sie ist ein Würfel aus Bildern, in dem sich alles abspielt. In diesem Fall sind die Bilder einfach ein paar weit entfernte scheinende Sterne.

Das Spielerraumschiff (PlayerShip)

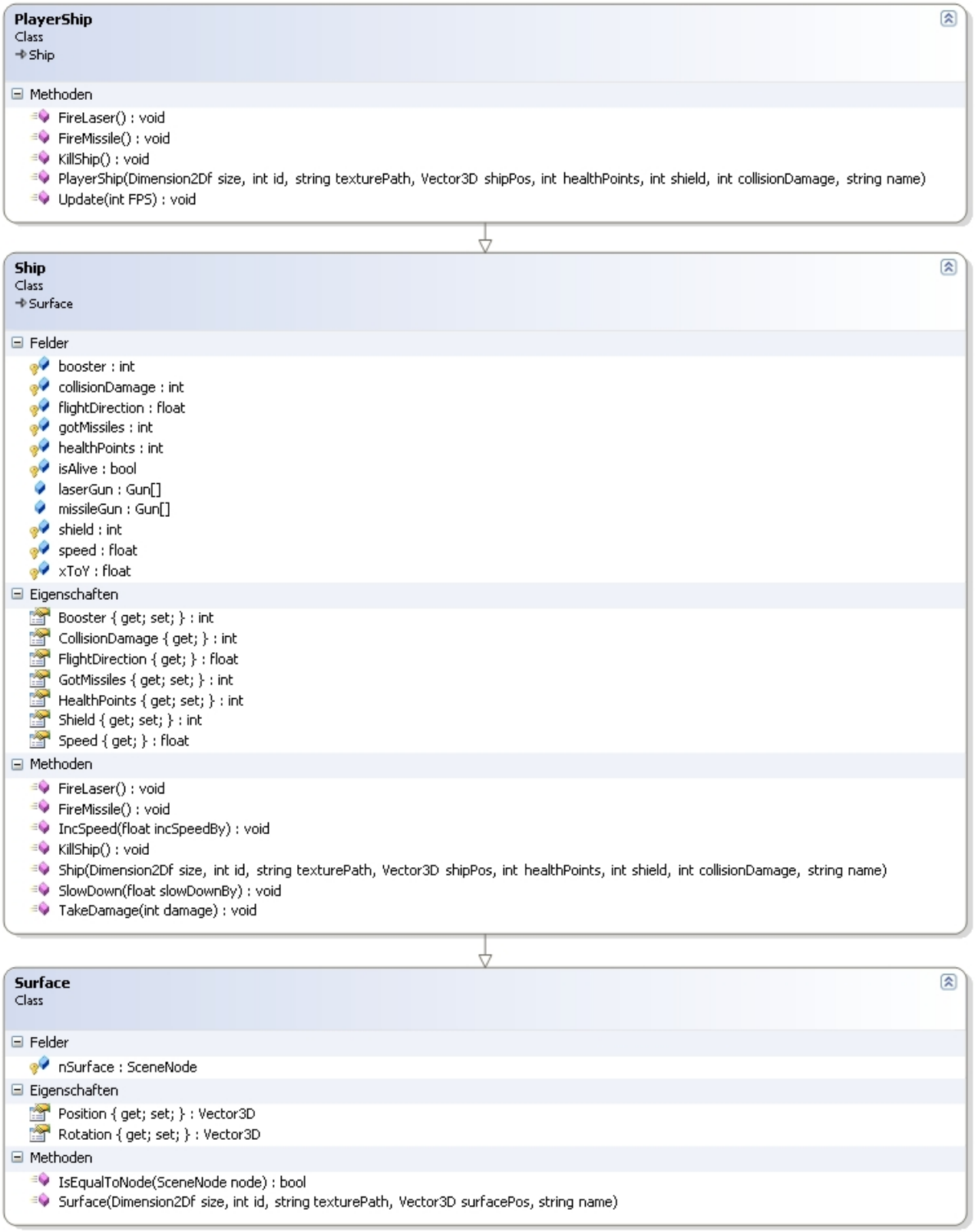

Abbildung 16: Klasse PlayerShip, Klasse Ship, Klasse Surface (Vererbungsbaum)

Um die Eigenheiten des Spielerraumschiffes genauer zu verstehen, müssen erst die beiden Basisklassen betrachtet werden.

Surface:
Surface ist die Basisklasse mehrerer Klassen in diesem Projekt. Sie ist eine 2D-Fläche die im Raum verschoben, skaliert und rotiert werden kann. Auf diese Fläche kann eine Textur aufgezogen werden. So ist z.B. ein Spielfeldrahmenstück des Border Managers nichts anderes als ein Objekt dieser Klasse.

Ship:
Ein Raumschiff (bezogen auf dieses Spiel) kann hauptsächlich fliegen und schießen. Was genauer beim Abfeuern eines Lasers passiert, wird im Kapitel „Abfeuern eines Schusses“ erläutert.
Für das Fliegen wurde ein eigenes Prinzip entwickelt. Es basiert auf zwei 2D Vektoren.

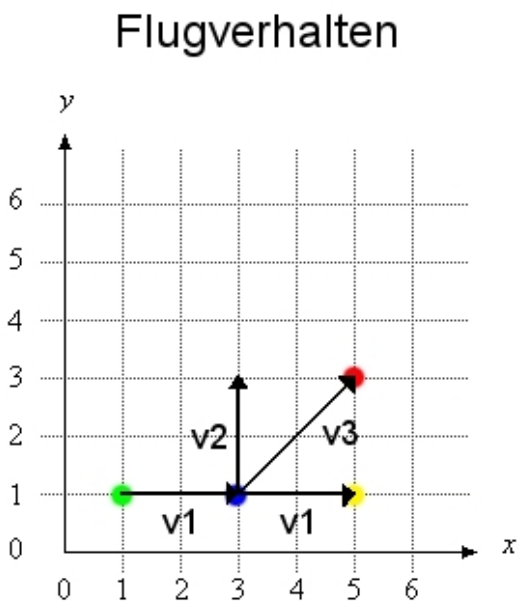

Abbildung 17: Flugverhalten eines Raumschiffes

Das Raumschiff befindet sich an Punkt **A(1|1)** und fliegt mit Richtung 0 (entlang der X-Achse) und einer Geschwindigkeit von 2.

Nach einem Durchlauf ist das Raumschiff an Punkt **B(3|1)**. Würde nichts weiter passieren, so ist es nach einem weiterem Durchlauf am Punkt **C(5|1)**.

Wenn jedoch in Punkt **B** das Raumschiff seine Richtung um 90° wechselt und in die neue Richtung mit 2 beschleunigt, so kommt es am Punkt **D(5|3)** an.

Die Rechnung hierfür ist recht einfach, da ein simples Vektorenaddieren ausreicht.

v3 = v1 + v2

v3 wird dann selbstverständlich zum neuen v1.

Die Werte des ersten Vektors bestehen somit aus Richtung und Geschwindigkeit und die des zweiten Vektors aus neuer Richtung und Beschleunigung.

v1(Richtung | Geschwindigkeit)
v2(neue Richtung | Beschleunigung)

Die jeweils neue Position lässt sich also aus den beiden Vektoren berechnen als:

Neue Position X = Alte Position X + aus v1 X und v2 X errechneten v3 X
Neue Position Y = Alte Position Y – aus v1 Y und v2 Y errechneten v3 Y

Da es sich bei der Richtung um eine Zahl in Bogenmaß handelt und auch die Geschwindigkeit keine Koordinatenangabe ist, ist die Praxis etwas komplexer. Hier müssen erst einige trigonometrische Berechnungen durchgeführt werden. Erst dann ist es möglich v3, und somit die neue Position des Schiffes zu berechnen[11].

11) Quellcodereferenz: Datei „ShipSimilarities.cs“, Funktion „IncSpeed“ und Datei „PlayerShip.cs“ Funktion „Update“

PlayerShip:
Da die Basisklasse Ship für das Spielerraumschiff fast alle Funktionen bietet, sind nur noch kleinere Änderungen nötig. So wird beispielsweise die Methode FireLaser() lediglich so angepasst, dass der Laser grün ist und von der Spitze des Raumschiffes los fliegt.

Die einzig neue Funktion der Klasse, Update(int FPS), ist komplexer. In ihr wird die neue Position des Schiffes errechnet, sowie der Kollisionstest, ob das Schiff mit einem anderen oder mit der Spielfeldgrenze kollidiert. Auch die Kollisionserkennung, ob einer der abgefeuerten Schüsse ein Ziel trifft, wird hier aufgerufen.

Der Powerup Manager (PowerUpMgr)

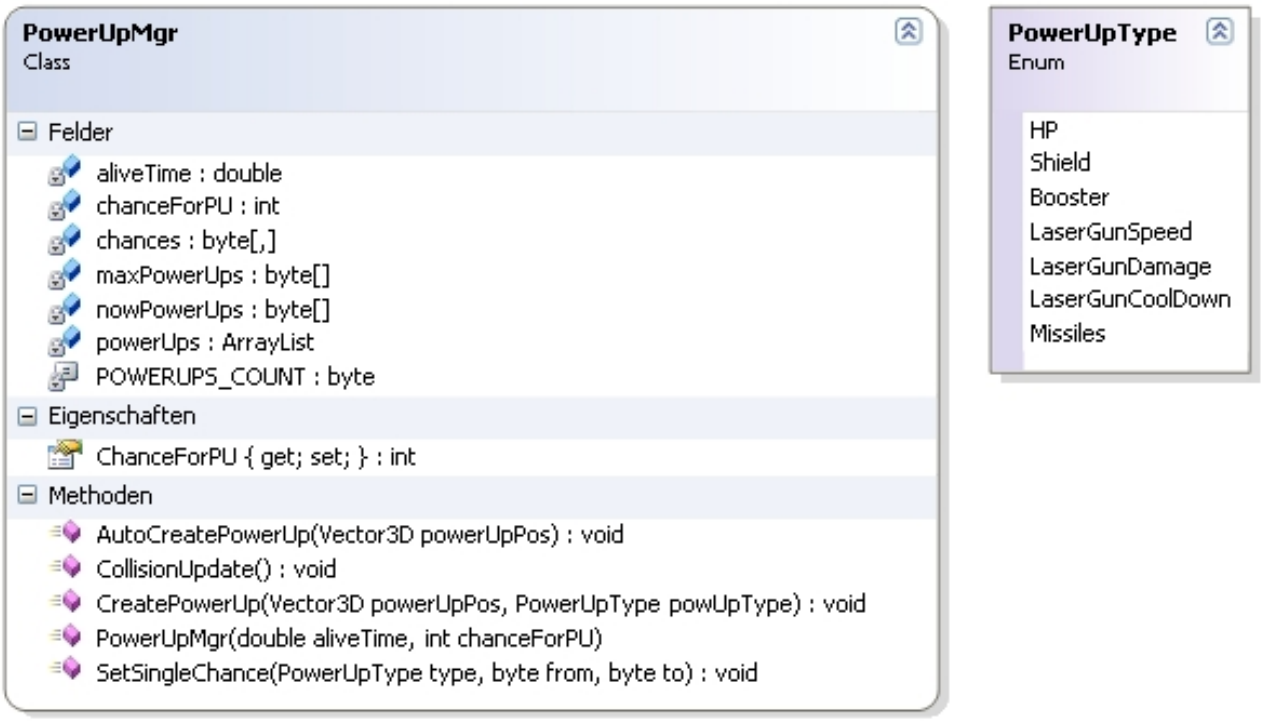

Abbildung 18: Links: Klasse PowerUpMgr, Rechts: Enumeration PowerUpType

Der Powerup Manager ist für die Erstellung von Powerups zuständig. Wenn ein Raumschiff zerstört wird, so besteht eine definierte Wahrscheinlichkeit, ob es ein Powerup hinterlässt oder nicht. Ist die Entscheidung gefallen ein Powerup zu kreieren, gibt es wieder eine Wahrscheinlichkeit, mit der ein Powerup eines bestimmten Typs erstellt wird.

Beispiel:
Allgemeine Erstellungswahrscheinlichkeit: 1:10 (Zufallszahl zwischen 1 und 10. Wenn Zufallszahl 1, dann wird ein Powerup erstellt.)
Wahrscheinlichkeit für einen einzelnen Typ:

HP	Shield	Booster	L.G.Speed	L.G.Damage	L.G.CoolDown	Missiles
1 bis 20	21 bis 40	41 bis 50	51 bis 60	61 bis 70	71 bis 80	81 bis 100

(Zufallszahl zwischen 1 und 100 wird erstellt. Je nachdem in welchem Bereich sie liegt, wird passendes Powerup erzeugt.)

Alle Wahrscheinlichkeiten (allgemeine und für einzelne Typen) können verändert werden.

Eine weitere Besonderheit des Powerup Managers ist, dass dieser auch die Kollisionserkennung für die Powerups regelt. Wenn also ein Powerup mit dem Spielerraumschiff kollidiert, so verändert der Powerup Manager die Eigenschaften des Raumschiffes. Dies mag von der Denkweise paradox erscheinen, ist aber für die Programmierung leichter realisierbar.

Powerup Typen:

Grafik	Beschreibung
	Lebensenergie: Gibt dem Spieler ein Bonus von 50 HP (Lebenspunkten)
	Schildenergie: Gibt dem Spieler ein Bonus von 50 SP (Schildpunkten)
	Booster: Gibt dem Spieler die Möglichkeit schneller zu fliegen
	Laser Schaden: Die Schadenskraft der Laserkanone wird verstärkt
	Laser Geschwindigkeit: Der Laserstrahl der Kanone erreicht schneller sein Ziel
	Laser Feuerrate: Laserkanone kann schneller hintereinander schießen (CoolDown wird kleiner)
	Raketen: Das Spielerraumschiff wird mit zusätzlich mit zehn Raketen ausgestattet

Tabelle 5: Verschiedene Powerup Typen der Enumeration

Der Ship Manager (ShipMgr)

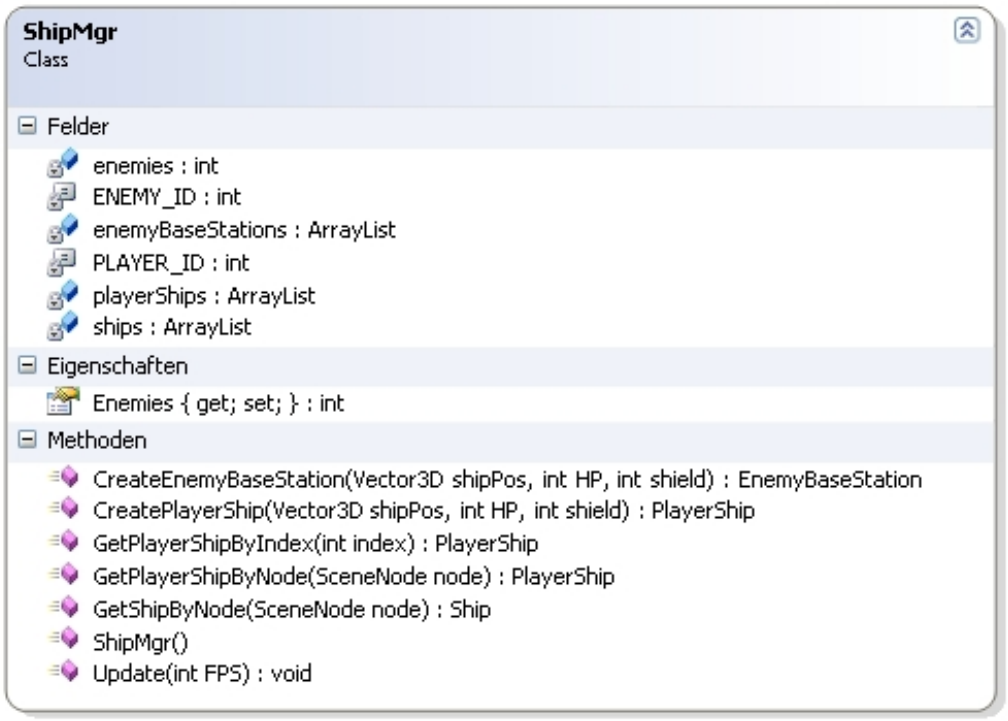

Abbildung 19: Klasse ShipMgr

Die Klasse ShipMgr ist für die Erstellung von Raumschiffen verantwortlich. Sie vereinfacht in erste Linie diese Erstellung, indem sie eine Norm für die Schiffe setzt. Das heißt, dass beispielsweise das Spielerraumschiff bis zu diesem Punkt noch voll dynamisch ist. Es steht nicht fest wie groß es ist, welche Grafik für es verwendet werden soll, oder wieviel HP es hat. Die Klasse setzt jetzt für einige dieser Eigenschaften ein Standard. So ist das Spielerraumschiff immer gleich groß und besitzt immer die gleiche Grafik. Werte wie Position, HP und Shield bleiben allerdings auch weiterhin dynamisch.

Weitere wichtige Funktion ist GetShipByNode(SceneNode node).
Der Ship Manager weiß von jedem erstellten Raumschiff. Wenn nun eine Kollision zwischen Spielerschuss und Gegnerraumschiff auftritt, kann als erstes lediglich ermittelt werden, welcher SceneNode (welches Szenen Mitglied) getroffen wurde. Über die Funktion GetShipByNode(SceneNode node) kann dann ermittelt werden, welchem Raumschiff der Schaden anzurechnen ist.

Der Space Manager (SpaceMgr)

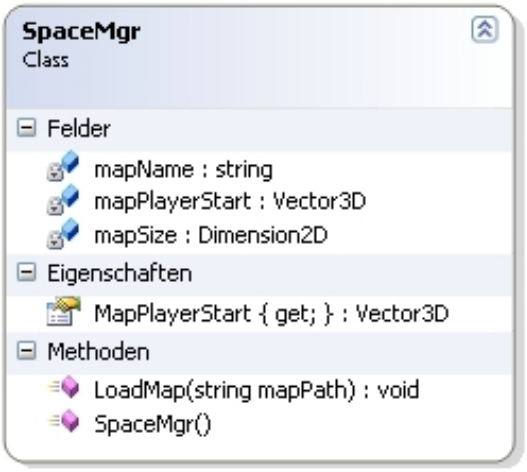

Abbildung 20: Klasse SpaceMgr

Trotz des kleinen Klassendiagramms ist der Space Manager eine der wichtigsten und größten Klassen. Er erstellt den kompletten Weltraum anhand einer Karte. Er initialisiert Schlüsselobjekte, wie den Border Manager, den Star Manager, den Dust Manager und den Galaxy Manager. Die ganzen Werte dafür entnimmt er einer externen Datei, der Karte. Diese ist in XML[12] geschrieben. Die Struktur, die Hierarchie sowie die Tags sind dabei selbst entworfen.

Aufbau der Kartendatei:

```
01 <map name="Test Map" width="13" height="13">
02   <galaxies>
03     <galaxy width="800" height="800">
04       <path>gfx/environment/galaxy5.jpg</path>
05       <position x="0" y="200" z="600"/>
06     </galaxy>
07   </galaxies>
08
09   <layers borderbuffer="100">
10     <layer index="0">
11       <maxstars>10</maxstars>
12     </layer>
13   </layers>
14
15   <player>
16     <position x="0" y="0"/>
17   </player>
18 </map>
```

12) „Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten.“ (Quelle: http://de.wikipedia.org/wiki/Extensible_Markup_Language)

Zeile	Beschreibung
1 und 18	Map-Tag: Umschließt alles. Attribute: Kartenname, und Kartengröße.
2 und 7	Galaxies-Tag: Bereich in dem einzelne Galaxien definiert werden können.
3 und 6	Galaxy-Tag: Definition einer Galaxie. Attribute: Größe.
4	Path-Tag: Pfad zu der Galaxie-Grafik.
5	Position-Tag: Position der Galaxie. Attribute: X, Y und Z Koordinaten.
9 und 13	Layers-Tag: Bereich in dem Layerinformationen definiert werden. Attribut: Pufferbereich in dem Sterne noch nicht neu positioniert werden sollen.
10 und 12	Layer-Tag: Definition zu einem der drei Layer. Attribut: Index des Layers (0-2)
11	Maxstars-Tag: Gibt an, wie viele Sterne in dieser Ebene sein sollen.
15 und 17	Player-Tag: Bereich für die Definition von Spielereigenschaften.
16	Position-Tag: Position des Spielerraumschiffs beim Start dieser Karte.

Tabelle 6: Syntaxerklärung zum Aufbau der Kartendatei

Informationen über das Schreiben und Lesen von XML sind im Kapitel „Funktionsweise des XML Schreibens / Lesens“ beschreiben.

Der Star Manager (StarMgr)

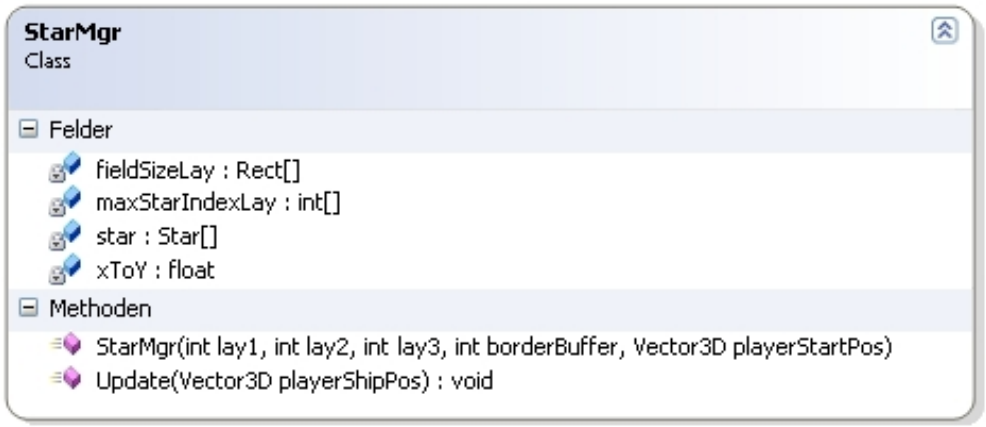

Abbildung 21: Klasse StarMgr

Die Klasse StarMgr ist für die Positionierung der Sterne im Weltraum zuständig. Die Sterne sind dabei in drei Ebenen aufgeteilt. Dabei hat jede der Ebenen eine andere Grafik für die Sterne. So sehen weiter weg entfernte Sterne anders aus und ziehen langsamer. Wenn ein Stern über den Bildschirmrand plus den Puffer hinaus ist, so wird er auf der anderen Seite wieder an einer zufälligen Position (auf der gleichen Ebene) eingesetzt.

Klassenkommunikation

Die Kommunikation der Klassen ist ein wichtiger Bestandteil fast jeder objektorientierten Anwendung. Vergleichbar mit der Fließbandproduktion von Autos tut auch hier jedes Objekt seine Arbeit für sich korrekt erledigen. Doch das fertige Produkt (Auto) kann erst dann funktionieren wenn alle richtig miteinander arbeiten.

Die Kommunikation der Klassen untereinander tritt in diesem Projekt häufig bei deren Initialisierung auf. Dennoch findet auch ein großer Informationsaustausch während der normalen Spiellaufzeit statt. Das Beispiel „Laden eines Levels“ zeigt wie Informationen zur Erstellung von Objekten weitergegeben werden. Das andere Beispiel, „Abfeuern eines Schusses“, erläutert wie Objekte nach deren Initialisierung Informationen austauschen.

Laden eines Levels

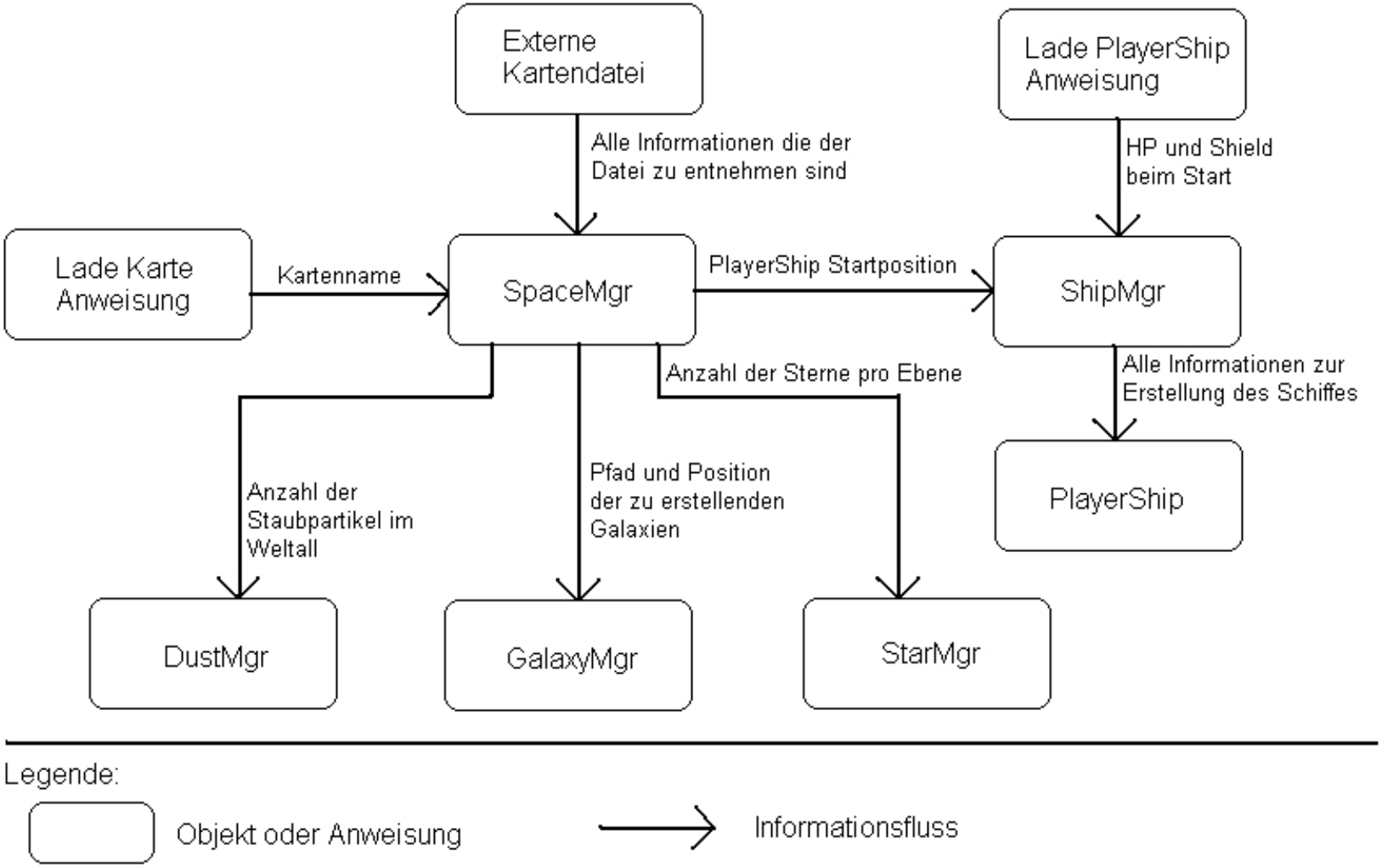

Abbildung 22: Informationsaustausch zwischen Objekten beim Laden eines Levels

Abfeuern eines Schusses

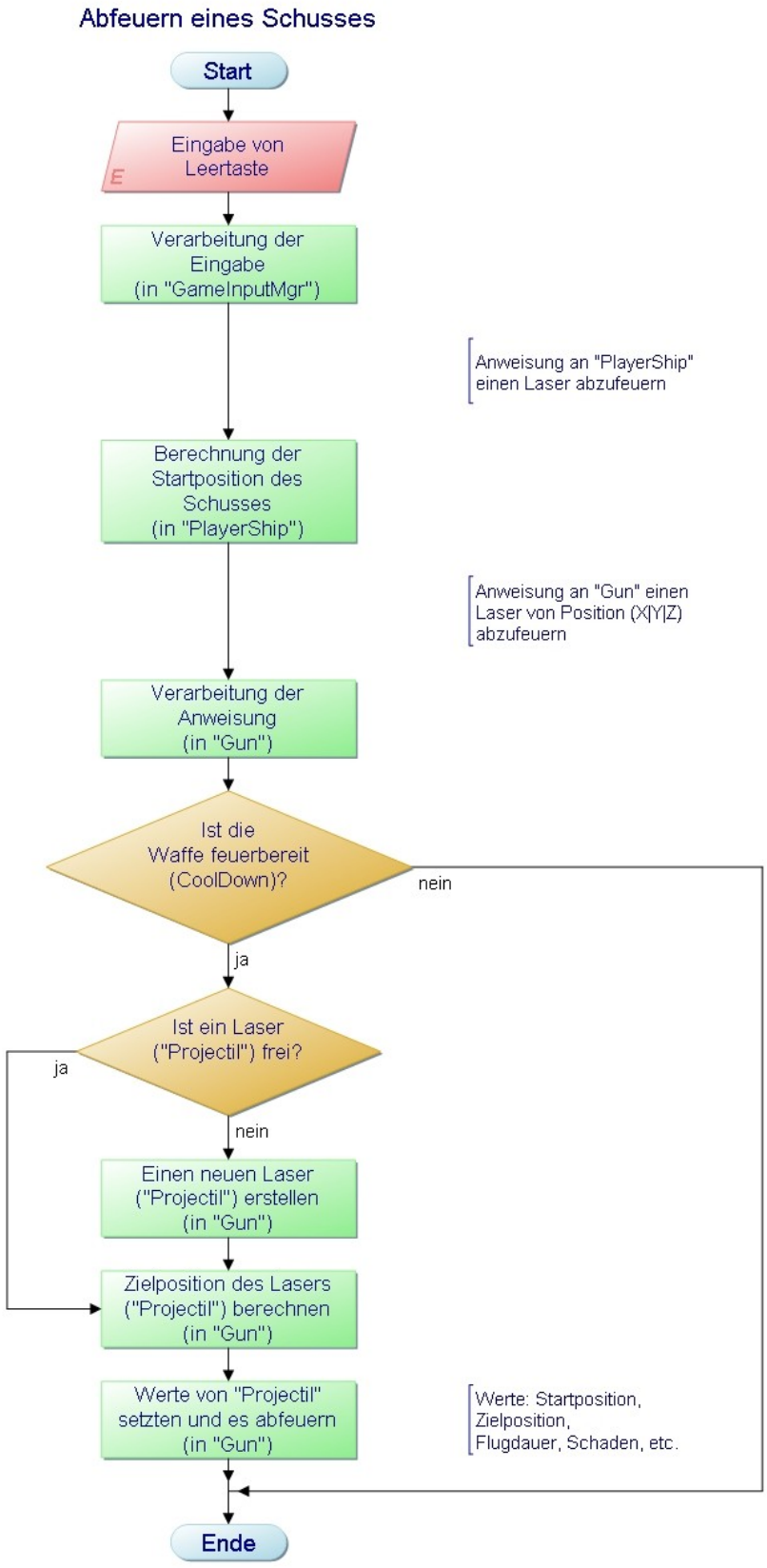

Abbildung 23: Informationsweitergabe beim Abfeuern eines Schusses

Der Schuss trifft

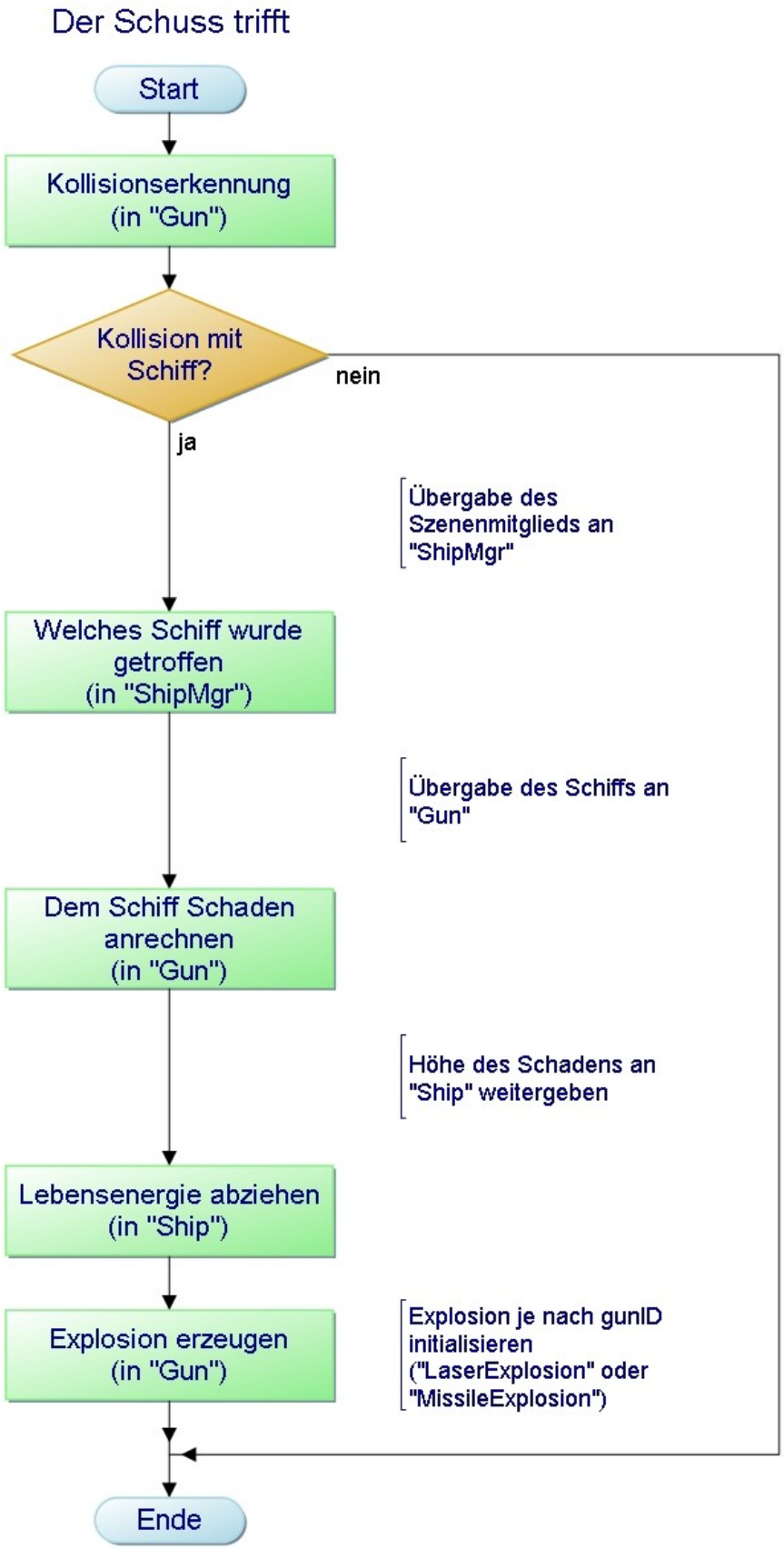

Abbildung 24: Informationsaustausch bei einem treffenden Schuss

Das Einstellungsprogramm

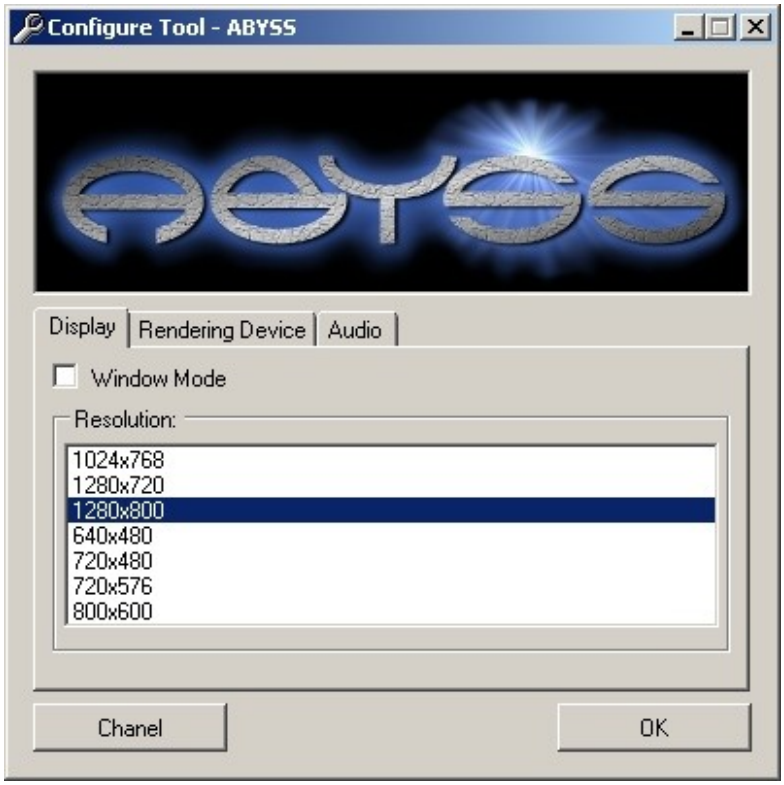

Abbildung 25: Oberfläche des Einstellungsprogramms

Dieses Programm wurde separat entwickelt, um das Spiel zu konfigurieren. Es handelt sich dabei um eine Anwendung in Form eines Windowsfensters. Mit ihr können die Auflösung, das Rendering Device[13] und die Audioeinstellungen festlegt werden.

Das Wesentliche an diesem Programm ist es eine XML Datei lesen und schreiben zu können. Auch diese Struktur, Hierarchie so wie die Tags sind selbst entworfen. Der genauere Aufbau wird im Kapitel „Aufbau der Einstellungsdatei“ beschrieben. Um als Entwickler vor dem Start des Spiels eine Auflösung und ein Rendering Device zu wählen, kann man die Option „-dev“ verwenden. (z.B. abyss.exe -dev)

Funktionsweise des XML Schreibens / Lesens

Beim XML-schreiben bzw. lesen wird schrittweise vorgegangen. Wenn beispielsweise die „Window Mode“-Einstellung ausgelesen werden soll, (siehe Kapitel „Aufbau der Einstellungsdatei“) so wird sich Schritt für Schritt daran heran gearbeitet. Erst wird nach dem Config-Tag gesucht. Ist dieser vorhanden, wird nach dem Display-Tag gesucht. Ist dieser auch vorhanden, wird nach dem Windowmode-Tag gesucht. Wenn letzten Endes dieser auch gefunden wurde, kann aus ihm die Einstellung entnommen werden.

Um mit den XML Dokumenten dementsprechend umzugehen, wurde der XML-Parser[14] von Microsoft verwendet und individuell angepasst.

13) Als Renderer bezeichnet man im Allgemeinen das „Gerät“, dass sich um die Darstellung oder Berechnung von grafischen Inhalten kümmert.

14) „Programme oder Programmteile, die XML-Daten auslesen, interpretieren und ggf. auf Gültigkeit prüfen, nennt man XML-Parser.“
(Quelle: http://de.wikipedia.org/wiki/Extensible_Markup_Language)

Aufbau der Einstellungsdatei

```
01 <config>
02   <display>
03     <windowmode>True</windowmode>
04     <resolution x="800", y="600" />
05   </dispaly>
06   <videodriver>
07     <driver>0</driver>
08   </videodriver>
09   <audio>
10     <music>False</music>
11     <effects>False</effects>
12   </audio>
13 </config>
```

Zeile	Beschreibung
1 und 13	Config-Tag: Umschließt die ganzen Einstellungen.
2 und 5	Display-Tag: Bereich, in dem bildschirmspezifisch Einstellungen gemacht werden.
3	Windowmode-Tag: Boolescher Wert. Legt fest, ob Spiel im Fenster oder Vollbild gestartet werden soll.
4	Resolution-Tag: Bildschirmauflösung. Attribute: Horizontale Pixelanzahl X, Vertikale Pixelanzahl Y.
6 und 8	Videodriver-Tag: Bereich, in dem Videotreiber bezogene Einstellungen vorgenommen werden.
7	Driver-Tag: Gerät mit dem Grafik dargestellt werden soll. (0=Direct3D 9c, 1=Direct3D 8.1, 2=OpenGL 1.5, 3= Software Renderer, 4=Apfelbaum Software Renderer)
9 und 12	Audio-Tag: Bereich, in dem Audioeinstellungen getroffen werden.
10	Music-Tag: Boolescher Wert. Musik an / aus. (in dieser Version noch nicht unterstützt)
11	Effects-Tag: Boolescher Wert. Soundeffekte an / aus. (in dieser Version noch nicht unterstützt)

Tabelle 7: Syntaxerklärung der XML-Einstellungsdatei

Kostenkalkulation

Es wird von einem Stundenlohn in Höhe von 50€ ausgegangen.

Tätigkeit	Stunden	Preis
C# lernen	15	750,00 €
Umgang mit Irrlicht-Engine lernen	20	1.000,00 €
Umgang mit XML lernen	8	400,00 €
Programmieren: Spiel	400	20.000,00 €
Programmieren: Einstellungsprogramm	50	2.500,00 €
Bildbearbeitung	20	1.000,00 €
Summe	**513**	**25.650,00 €**

Tabelle 8: Kalkulation der Projektkosten

Schlusswort

Es ist nahe liegend dass ein rund 3800 Zeilen (61 Din A4 Seiten) umfassendes Projekt nicht ins kleinste Detail in dieser Dokumentation beschrieben ist. Da im Quellcode aber jede Klasse sowie fast jede Funktion mit Kommentar (Englisch) versehen ist, sollte es möglich sein ein detaillierteres Verständnis zu erlangen.

Das Projekt bietet in diesem Stadium eine breite Basis.
Trotz kleiner Fehler und „Baustellen“, ist es schon als Spiel einsatzfähig. Des weiteren ist es leicht erweiterbar. Durch das Vererbungssystem ist das Erstellen von neuen Gegnerraumschiffen, Powerups, Waffen und Leveln mit überschaubarem Zeitaufwand möglich. Auch die Vertonung ist mit dem Einbinden einer fremden Sound-Engine wie „IrrKlang“, eine einfache Erweiterung.

Baustellen

- Flamme für die Treibwerksdüse des Spielerraumschiffs
- Drehen der Staubpartikel um ihre Spitze und nicht um ihren Mittelpunkt
- Menüpunkt „Introductions“ und „Credis“ gestalten
- Spielpause, wenn Spieler im Menü ist

Bekannte Fehler

Leider ist die Grafik-Engine mit der Version 1.3.1 nicht in der neusten Version.
(Aktelle Version: Irrlicht 1.5.0, Stand 26.05.09). Es treten deswegen hin und wieder Fehler auf, die in neueren Versionen schon behoben sind.

- Wenn der Spieler verliert / gewinnt sollte das Spiel neu gestartet werden, da sonst , wenn man abermals den Menüeintrag „Start Game“ wählt, ein Engine-spezifischer Fehler auftritt.

Quellenangabe

- Wikipedia
  - http://de.wikipedia.org/wiki/Powerup
  - http://de.wikipedia.org/wiki/Grafik-Engine
  - http://de.wikipedia.org/wiki/Open_Source
  - http://de.wikipedia.org/wiki/Online-Community
  - http://de.wikipedia.org/wiki/Wrapper_(Software)
  - http://de.wikipedia.org/wiki/Mesh
  - http://de.wikipedia.org/wiki/Textur_(Computergrafik)
  - http://de.wikipedia.org/wiki/Head-Up-Display
  - http://de.wikipedia.org/wiki/Bildfrequenz
  - http://de.wikipedia.org/wiki/Extensible_Markup_Language

- Irrlicht
  - http://irrlicht.sourceforge.net/
  - http://irrlichtnetcp.sourceforge.net/wiki/index.php/Main_Page
  - http://irrlicht.sourceforge.net/phpBB2/index.php

- Design / Medienquellen
  - http://cooltext.com/
  - http://www.gimp.org/
  - http://www.turbosquid.com/
  - http://planetpixelemporium.com/
  - http://hubblesite.org/

- Bücher
  - Visual C# 2008, Galileo Computing, ISBN: 978-3-8362-1172-7
    (auch frei erhältlich als OpenBook unter
    http://www.galileocomputing.de/katalog/openbook)
  - Das Einsteigerseminar C# Objektorientierte Programmierung, bhv,
    ISBN: 3-8266-7228-3

Verwendete Software

- Microsoft Visual Studio 2008 (C#)
- Irrlicht Engine SDK 1.5.0 (Tools für die Engine)
- Irrlicht.NET CP SDK 1.3.1 (Engine mit Wrapper)
- GIMP (Bildbearbeitung)
- OpenOffice (Writer)
- PapDesigner (Programmablaufplaner)
- Microsoft .NET Framework 3.5 (Projektabhängigkeit)
- Microsoft DirectX 9c (Projektabhängigkeit)

Inhalt der beiliegenden CD

- Das Spiel als Setup Paket (Binary)
- Das Spiel als Visual Studio 2008 Projekt (Source Code)
- Das Einstellungsprogramm als Visual Studio 2008 Projekt (Source Code)
- Microsoft .NET Framework 3.5 (Projektabhängigkeit)
- Microsoft DirectX 9c (Projektabhängigkeit)
- Diese Dokumentation als PDF