Starjumper

Robert Strobl, Johannes Albrecht, Tim Berning, Stefan Härtel

Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam

1 Einleitung

1.1 Spielprinzip

Starjumper ist ein Renn- und Geschicklichkeitsspiel, bei dem der Spieler mit einem Raumschiff über eine hindernisreiche Strecke mit diversen Abgründen fahren und schlussendlich sicher ins Ziel gelangen muss.
Dabei besitzen die Flächen der Streckenabschnitte unterschiedliche Eigenschaften, die sich auf das Fahrverhalten des Raumschiffs auswirken können, z.B. Abbremsen des Spielers.
Die Steuerung umfasst Beschleunigen, Bremsen, Lenken nach links oder rechts und Springen.

1.2 Spielidee

Vorlage für unser Spiel war der Spieleklassiker Skyroads aus dem Jahr 1993, welches von Bluemoon entwickelt wurde und wiederum selbst eine Neuauflage des Spiels Kosmonaut aus dem eigenen Hause darstellte.
Das Spielprinzip wurde für Starjumper größtenteils übernommen.

2 Steuerung

Die Steuerung des Raumschiffs erfolgt sehr intuitiv, der Spieler kann beschleunigen, bremsen, lenken und springen. Die folgende Tabelle zeigt die in-game Tastaturbelegung:

Beschleunigen	Pfeiltaste oben
Bremsen	Pfeiltaste unten
Links	Pfeiltaste links
Rechts	Pfeiltaste rechts
Springen	Leertaste

Table 1. Tastaturbelegung

Darüber hinaus kann per `Escape` jederzeit aus dem Spiel in das Menü bzw. in die jeweils nächsthöhere Ebene in der Menühierarchie gewechselt werden. Im Hauptmenü beendet ein Druck auf `Escape` das Programm.
Die Navigation in den Menüs erfolgt mithilfe der Maus.

3 Architektur

Die Beschreibung der Architektur erfolgt in kleineren logischen Einheiten, eine Gesamtübersicht aller verwendeten Klassen und ihrer Verbindungen untereinander findet sich im Anhang.
Im Folgenden sollen nun die wichtigsten architektonischen Bestandteile von Starjumper erläutert werden.

3.1 GameManager

Der `GameManager` bildet die oberste Instanz des Spiels. Er verwaltet primär den `Viewer`, der für das Rendering des Szenengraphen verantwortlich ist, sowie die Szenengraphen (in Form von `RenderingInstance` Objekten), zwischen denen er dynamisch umschalten kann. Die Klasse `RenderingInstance` ist ein Wrapper für Szenengraphen und dient als abstrakte Oberklasse für u.a. `Game` und `Menu`, die später erläutert werden.
Darüber hinaus implementiert der `GameManager` übergeordnete Spiellogik wie das Starten des Spiels, Beenden, Erstellen und Aufrufen von Menüs, sowie die Bereitstellung von Callback-Funktionen für die Menüs. Ein übergeordneter Keyboardhandler sorgt für die globale Logik der Escape Taste.

3.2 Menu

Die Aufgabe der Klasse `Menu` ist das Erstellen eines Menüs mit klickbaren Textelementen. Diese werden als Callback auf entsprechende Methoden des `GameManagers` gemappt. Zur Anzeige der Menüeinträge kommt das Modul osgWidget zum Einsatz.

3.3 Game

Die von `RenderingInstance` erbende Klasse `Game` beinhaltet die zentrale Spiellogik. Von hier aus werden die für das Spiel benötigten Elemente wie etwa `HeadUpDisplay`, `Player` und `Level` erzeugt und verwaltet, sowie der schlußendlich zu rendernde Szenengraph generiert.
Neben den für OpenSceneGraph wichtigen Komponenten verwaltet `Game` auch die Bullet Physics World und sorgt für das OSG-synchrone Stepping derselben mittels eines NodeCallbacks namens `WorldUpdater`.

3.4 Level

Starjumper verwendet ein einfaches XML-Format für die Beschreibung der Levels. Für das Parsen einer solchen Leveldatei und dem anschließenden Umwandeln in OSG-kompatible Form zeichnet `Level` verantwortlich.
Derzeit mögliche Elemente eines Levels sind Quader, Tunnel und das Ziel, entsprechend erzeugt `Level` jeweils eine Instanz der Klasse `Cuboid`, `Tunnel` oder `Goal` und fügt sie in einer osg::Group zusammen.
Diese Elementklassen wiederum generieren jeweils eine Repräsentation für OSG und Bullet.

3.5 Player

`Player` stellt die vom Benutzer steuerbare Entität dar in Form eines Raumschiffs, das fix aus einer Datei geladen wird. Zwecks Positionierbarkeit wird das Model in einer PositionAttitudeTransform gekapselt. Neben dem Laden dieser Modeldatei werden in `Player` die Physikkomponenten für den Spieler generiert. Die Interaktion des Benutzers mit dem Spiel erfolgt über die Komponenten `PlayerState` und `PlayerUpdater`, die im nächsten Abschnitt näher erläutert werden.
Details zu der Kombination aus `KinematicCharacterController` und `btGhostObject` finden sich im Abschnitt "Designentscheidungen".

3.6 PlayerUpdater

Der `PlayerUpdater` als NodeCallback nutzt den `PlayerState`, um pro Step die neue Position des Players zu berechnen. Genauer wird diese aus der Physics Worlds extrahiert, die die Bewegung des Spielers simuliert und so eventuelle Kollisionen erkennen kann. Gleichzeitig werden diese Parameter aktualisiert, dies geschieht auf Basis von Anfragen, die der Benutzer über einen Tastendruck absetzt und die im `PlayerState` vermerkt werden.
Folgendes Sequenzdiagramm soll das Vorgehen des `PlayerUpdater`s veranschaulichen:

4 Designentscheidungen

4.1 Levelfiles

4.2 Physik

5 Design Patterns

6 Diskussion und Ausblick

A Vollständiges Klassendiagramm