Reaktive Programmierung SoSe 17

Christoph Lüth
Martin Ring

4. Übungsblatt

Ausgabe: 24.05.17
Abgabe: 08.06.17

4.1 *Curried in Space* *20 Punkte*

Raumfahrt ist auch 23. Jahrhundert noch ein Abenteuer, und Spaceman Spiff ist in einen unbekannten Teil des Weltraums vorgedrungen, den noch kein Mensch zuvor gesehen hat. In einem gefährlichen Asteroidengürtel gefangen, von den gräßlichen Schleimmonstern vom Planeten Zorg bedrängt, nur mit seinem Disruptor ausgerüstet — kann er sich retten?

Abbildung 1 zeigt Spiff in seiner Bedrängnis. Wir kommen Spiff mit unseren bescheidenen Mitteln zur Hilfe, und implementieren eine Simulation, die es erlaubt, den Weg nach Hause zu finden. Unsere Simulation ist (wie immer) Client/Server-basiert. Der Zustand des Weltraums mit allen seinen Objekten (Spiff und sein Raumschiff, Asteroiden, und Aliens) wird auf dem Server verwaltet. Der Client visualisiert lediglich den Zustand, und leitet Steuerkommandos an den Server weiter.

Bei der Implementierung unser Weltraumsimulation gehen wir in vier Schritten vor. Laden Sie zuerst die Vorgabe von der Webseite herunter. Diese enthält den Client, sowie einen rudimentären Server. Der Client visualisiert alle Objekte, die im Weltraum zu finden sind.

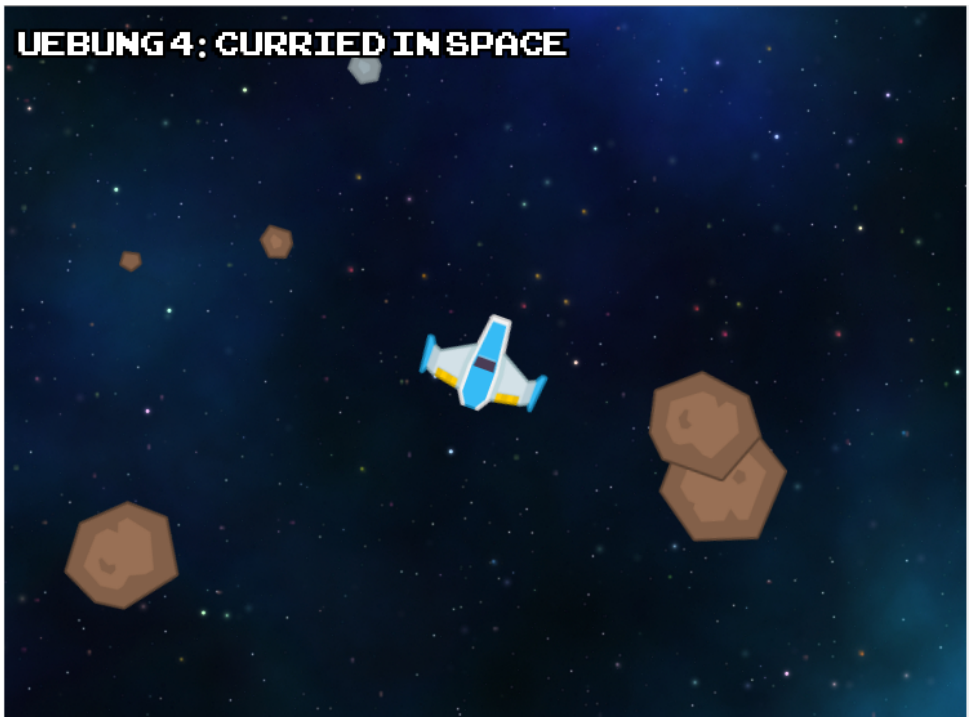

Abbildung 1: Lost in Space? Spaceman Spiff (Bildmitte) sucht den Weg nach Hause…

Die Objekte im Weltraum umfassen initial Spiffs Raumschiff, die Asteroiden, sowie die von Spiffs Disruptor abgefeuerten Photonenpakete. Ein Objekt ist durch folgende Attribute charakterisiert:

- Eine Position (als Punkt mit zwei Koordinaten (p_x, p_y), gemessen in Picoparsec (px)[1]);
- die Geschwindigkeit als Vektor (v_x, v_y) in px/s
- eine Beschleunigung a (als Skalar in px/s^2);
- eine Orientierung ω (als Bogenmaß, in Vielfachen von π);
- eine Winkelgeschwindigkeit ϕ (in Bogenmaß pro s).

Bei einem Zustandsübergang von δ Sekunden berechnen sich diese Attribute wie folgt neu:

$$
\begin{aligned}
(p'_x, p'_y) &= (p_x + v_x \cdot \delta, p_y + v_y \cdot \delta) \\
(v'_x, v'_y) &= (v_x - \delta \cdot \sin(\omega) \cdot a, v_y + \delta \cdot \cos(\omega) \cdot a) \\
\omega' &= \omega + \delta * \phi
\end{aligned}
$$

Die Beschleunigung a und die Winkelgeschwindigkeit ϕ ändern sich beim Zustandsübergang nicht; sie werden für das Raumschiff durch die Steuerkommandos gesetzt, und für die anderen Objekte durch ihren Kontrollalgorithmus berechnet (für die Asteroiden ist beispielsweise die Beschleunigung $a = 0$ und ϕ konstant; die Photonenpakete beginnen mit der Position, Orientierung und Geschwindigkeit des Raumschiffs, a und ϕ sind konstant, aber sie verschwinden sie nach einiger Zeit wieder).

1. Der Client berechnet einen solchen Zustandsübergang, und realisiert damit die vorgegebene Visualisierung. Im ersten Aufgabenteil wollen wir diesen Zustandsübergang auch im Server implementieren, um ihn dann im Server zu erweitern zu können. Der Server kann in Haskell oder Scala geschrieben werden.

   Auf dem Server wird jedes Objekt von einem Aktor verwaltet, der auf Anfrage den neuen Zustand des Objektes berechnet. Implementieren Sie dazu die nötigen Klassen oder Datentypen.

   Der gesamte Weltraum soll von einem Aktorsystem verwaltet werden, welches zum einen den Gesamtzustand neu berechnet, und zum anderen die Verbindung mit dem Client verwaltet. Die Verbindung mit dem Client wird von einem Aktor verwaltet, der regelmäßig einen neuen Zustand an den Client schickt. Implementieren Sie dazu eine Serialisierung, mit der Sie den Zustand von Objekten vom Server zum Client schicken können, und Steuerkommandos vom Client zurück an den Server.

   Um eine flüssige Darstellung der Objekte auch bei langsamer Netzverbindung (Spacemann Spiff ist schließlich am Rand des uns bekannten Universums, wo das Internet nicht mehr allzu schnell ist[2]) zu gewährleisten, werden die Nachrichten mit einem Zeitstempel versehen, so dass auf dem Client ein neuer Serverzustand zu einem gegeben Zeitpunkt einen neuen Clientzustand erzeugt. Beim Aufbau der Verbindung wird die Zeitverzögerung zwischen Client und Server berechnet.

   5 Punkte

2. Im zweiten Schritt soll jetzt die Interaktion der Objekte implementiert werden. Wenn Spiff gegen einen Asteroiden prallt, verliert Spiff ein Raumschiff (und wenn alle Raumschiffe zerstört sind, ist die Simulation beendet). Wenn ein Photonenpaket auf einen Asteroiden trifft, wird dieser in kleinere Asteroiden zerlegt; die kleinsten Asteroiden verlieren ihre intrinsische Feldspannung und lösen sich in ihre subatomaren Bestandteile auf. Die hierzu nötige Kollisionsberechnung können Sie approximativ handhaben, indem sie davon ausgehen, dass Asteroiden und das Raumschiff kreisförmig sind, und die Photonpakete in einem Punkt konzentriert sind.

   5 Punkte

3. In einem dritten Schritt werden die Schleimmonster vom Planeten Zorg implementiert. Die Schleimmonster verfolgen Spiff und werden mit widerlichen Schleimpaketen (ähnlich wie Spiffs Photonenpakete). Erweitern Sie die Visualisierung um weitere Raumschiffe und die Schleimpakete (das Verzeichnis[3] `assets/images/` enthält verschiedene andere Raumschiffe, die Sie benutzen können), und die Zustandsberechnung um einen Kontrollalgorithmus für die Schleimmonster und ihre Schleimpakete, sowie natürlich die Interaktion dieser Objekte mit den schon existierenden.

   3 Punkte

[1] Oder in anderen Worten, Pixeln der Bildschirmauflösung

[2] Eine Folge der allgemeinen Relativitätstheorie.

[3] Die Bilder in diesem Verzeichnis stammen von Kenney Vleugels `http://www.kenney.nl/` und sind frei benutzbar.

4. In einem vierten Schritt können Freunde Spiff zu Hilfe eilen! Implementieren Sie dazu einen Mehrspielermodus: wenn sich ein weiterer Client mit dem Server verbindet, soll hierzu ein weiteres Raumschiff erzeugt werden, dass sich von den anderen unterscheidet (beispielsweise durch die Farbe), und dass von diesem Client gesteuert wird.

   7 Punkte

5. Sie können außerdem weitere Objekte oder Funktionalitäten hinzufügen, beispielsweise Dilithiumpakete, mit welchen die Energie von Spiffs Disruptor wieder aufgeladen werden kann (dazu muss der Disruptor natürlich bei jedem Photonenpaket Energie verlieren), oder ein Schutzschild, der für eine kurze Zeit aktiviert werden kann (und dann natürlich Energie kostet).

   Bis zu 5 Bonuspunkte