WP Einführung in die Computergrafik

WS 2013/2014, Hochschule für Angewandte Wissenschaften (HAW), Hamburg
Prof. Dr. Philipp Jenke

Für alle praktischen Aufgaben gelten folgende Regeln:

- Der Code muss getestet sein (z.B. mit JUnit-Tests).
- Der Code muss die Code-Konventionen einhalten (finden Sie auf der EMIL-Seite).
- Die Funktionalität muss ausreichend kommentiert sein.
- Die Bonus-Aufgabe ist freiwillig und kann als Bonuspunkt bei der Klausur angerechnet werden

Aufgabenblatt 6

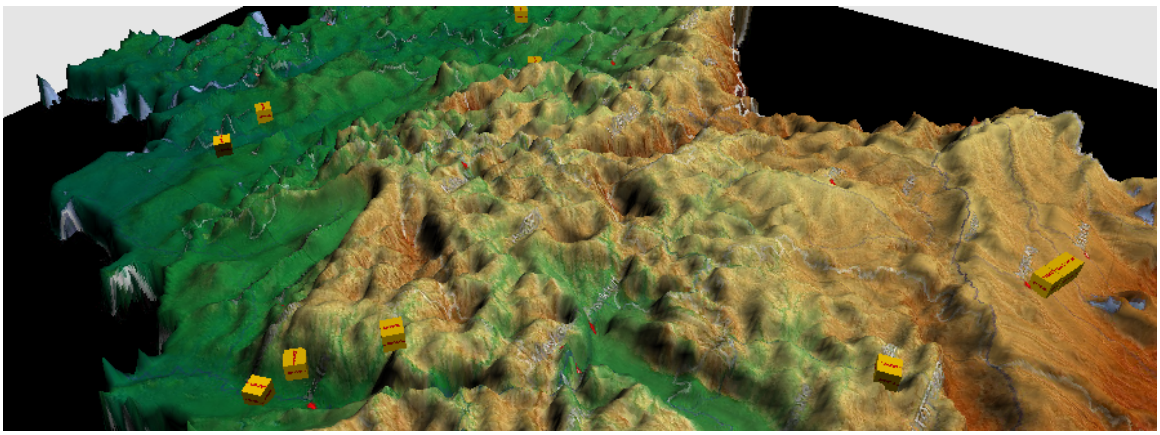

Abbildung 1: Visualisierung in der Musterlösung: die gelben Pakete werden gesteuert durch die HLS-Software zwischen Deutschen Großstädten hin- und hertransportiert.

Dieses Aufgabenblatt ist anders als die anderen. Es gibt hier keine Unterteilung in Theorie und Praxis. Außerdem fasst es die eigentlichen Blätter 6 und 7 zusammen. Es ist Ihnen freigestellt, zu welchem der beiden vorgesehenen Praktikumstermine Sie Ihre Ergebnisse vorstellen.

In diesem Aufgabenblatt entwickeln Sie eine Visualisierung für Transportaufträge, die in unserem Partner-WP *Softwareprojekt* verwaltet werden. Dort entsteht eine Logistikverwaltung mit dem Namen HLS. Die zentrale Herausforderung ist es, verschiedene Systeme, Technologien und Teilbereiche zu integrieren. So ist HLS in C# entwickelt, wir verwenden bekanntlich Java. Die Kommunikation wird über die Message-Queue RabbitMQ (*http://www.rabbitmq.com/)* abgewickelt. Die eigentlichen Nachrichten in der Kommunikation sind in JSON (*http://www.json.org/)* geschrieben.

Die Zusammenarbeit mit dem anderen WP bedeutet auch, dass sich Teams aus den beiden Veranstaltungen zugeordnet werden. Die Zuordnungen finden Sie rechtzeitig auf der EMIL-Seite. Ihrem zugeordneten Team können Sie Fragen zu der Kommunikation und zu RabbitMQ stellen. Gleichzeitig sollten Sie Anfragen zur Visualisierung beantworten. Da es im WP Computergrafik mehr Teams gibt, haben wir leider keine 1:1-Zuordnung. Beginnen Sie rechtzeitig mit der Umsetzung, damit Ihr Partnerteam auch ausreichend Zeit für Antworten hat.

Abbildung 2 beschreibt das zu entwickelnde System. Sie müssen allerdings nicht alle Komponenten neu entwickeln.

Vorgaben
Mit dem HLS-System müssen verschiedene Nachrichten ausgetauscht werden: Transportbeziehungen, Frachtauftrag und Sendungsereignis. Für diese Nachrichten gibt es bereits Klassen im Package `hls` (Transportbeziehungen → `Connection` und `Connections`, Frachtauftrag → `TransportOrder`, Sensungsereignis → `TransportEvent`). Diese Klassen liefern auch bereits Funktionalität, um entsprechende JSON-Nachrichten zu lesen und zu generieren (`fromJson` und `toJson`). Auch für die Kommunikation mit RabbitMQ gibt es bereits eine Klasse, die die eigentliche Funktionalität in einfache Methoden kapselt (Package `rabbitmq`, Klasse `RabbitMqCommunication`).

Informationen zu den Details der Kommunikation mit RabbitMQ finden Sie im beigefügten Dokument *CgCooperationDocument.pdf*.

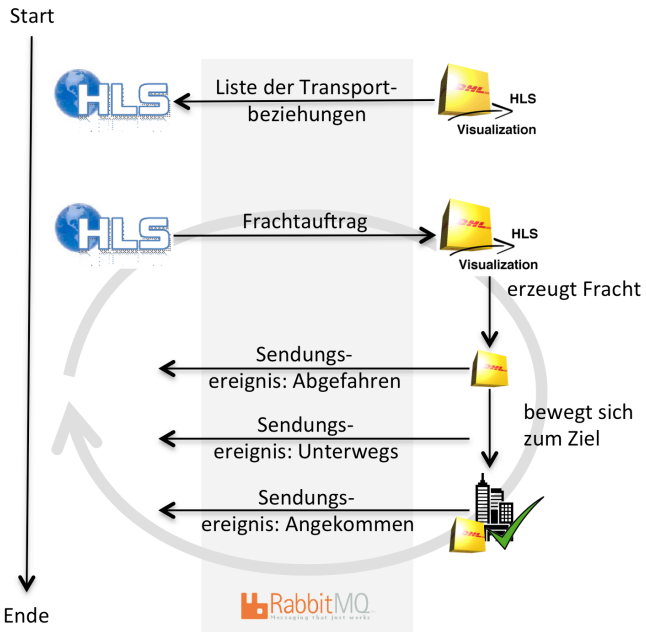

Abbildung 2: HLS-Visualisierung

Das Transportnetzwerk in dieser Simulation ist durch einen Graphen zwischen den zehn größten Städten Deutschlands repräsentiert. Auch diese Repräsentation ist bereits vorgegeben (Stadt → `City`, Graph zwischen den Städten → `TransportNetwork`). `TransportNetwork` ist eine statisch gehaltene Klasse, die den eigentlichen Graph als Singleton (http://de.wikipedia.org/wiki/Singleton_%28Entwurfsmuster%29) beinhaltet. Einige Hilfsklassen für den Umgang mit Graphen befinden sich im Package `graph`.

Als Vorgaben finden Sie außerdem einige Testklassen und die notwendigen Jar-Bibliotheken, um RabbitMQ und den JSON-Parser verwenden zu können. In dem Interface HlsConstants befinden sich einige Konstanten. Unter anderem können Sie dort auch die Namen der drei beteiligten Queues und weitere Parameter Ihrer Lösung ablegen.

Machen Sie sich zunächst ausgiebig mit diesen Vorgaben vertraut.

Anpassung des Frameworks
Im Vergleich zu den vorherigen Aufgaben, müssen Sie das Framework ein wenig anpassen. Verwenden Sie zunächst das Höhenfeld und die Farbkarte für Deutschland als Basis. Sorgen Sie außerdem dafür, dass in Ihrer Visualisierung jeder Timer-Aufruf (Methode `tick()` in der Frame-Klasse) einer Echtzeit entspricht. Jeder Tick soll für 5 Minuten stehen. Ihr Programm soll zu der Zeit

2014-12-08 00:00:00

starten. Unter Umständen sind dazu hier noch gar keine Änderungen notwendig. Die Pakete sollen sich später aber mit den richtigen Zeitschritten bewegen und zum richtigen Zeitpunkt starten.

Senden der Transportbeziehungen
Senden Sie bei Programmstart die Transportbeziehungen an die richtige RabbitMQ-Queue. Die Transportbeziehungen sind die Kanten im Graph.

Erzeugen eines Sendungsobjektes
Nun müssen Sie sich bei der Queue für Frachtaufträge registrieren. Das machen Sie mit einer Instanz von `RabbitMqCommunication` und indem Sie das Interface `IMessageCallback` implementieren. Wenn Sie einen Frachtauftrag erhalten, prüfen Sie zunächst, ob der Startzeitpunkt in der Zukunft liegt. Falls nicht, verwerfen Sie den Auftrag. Ansonsten merken Sie sich den Auftrag.
Aus der Liste der gemerkten Aufträge entnehmen Sie dann zum Startzeitpunkt (oder zum ersten Tick danach) den Auftrag und erzeugen ein neues `Movable`-Objekt. Aus dem Frachtauftrag können Sie die Startstadt und die Zielstadt entnehmen. Wahrscheinlich müssen Sie die Fortbewegungsmethode von `Movable` anpassen. Das

Movable soll natürlich exakt zur Zielzeit an der Zielstadt ankommen. Beispiel: Für eine Streckendauer von 60 Minuten zwischen zwei Städten soll das `Movable` genau 12 Timer-Ticks benötigen.

Sendungsereignisse
Jede Sendung informiert über seinen Zustand mit Hilfe von Sendungsereignissen. Sobald es startet, sendet es die Nachricht "Abgefahren". Auf dem Weg zu festen Zeitpunkten (z.B. alle 15 Minuten) sendet es eine Sendungsereignis-Nachricht vom Typ "Unterwegs" mit seinen aktuellen Koordinaten. Hat es sein Ziel erreicht, dann sendet es noch eine Nachricht vom Typ "Angekommen".

Entfernen von Sendungsobjekten
Hat die Sendung seine Zielstadt erreicht, dann wird das Objekt nicht mehr benötigt und wird aus dem Szenengraph entfernt.

Visualisierung
In der finalen Version soll Ihre Visualisierung mit einer HLS-Instanz Ihres Partnerteams laufen. Zur Entwicklung können Sie aber auch die Klasse `HlsSimulator` verwenden. Diese erzeugt ebenfalls Frachtaufträge in Queues und funktioniert ohne HLS. Der Simulation müssen Sie ebenfalls das Tick-Ereignis (zusammen mit der aktuellen Zeit) senden. Sie müssen wahrscheinlich die Namen der Queues entsprechend der Konventionen an Ihre eigenen Bedürfnisse anpassen.

Bonusaufgabe 1
Verwenden Sie sinnvolles texturiertes Objekt um eine Sendung darzustellen (z.B. einen LKW oder ein Paket).

Bonusaufgabe 2
Einige Pakete sollen per Flugzeug transportiert werden. In dem Fall bewegt sich das Paket nicht auf der Oberfläche des Höhenfeldes sondern auf einer sinnvollen Flugbahn vom Start zum Ziel (ohne Zwischenstädte, zunächst steigend, dann sinkend, am Besten mit glattem Übergang). Pakete auf dem Flugweg müssen natürlich auch durch ein geeignetes Objekt dargestellt werden (z.B: 3D-Modell eines Flugzeugs).

Ein Video der Musterlösung finden Sie hier:
http://users.informatik.haw-hamburg.de/~abo781/videos/hls.mp4

Hinweise: Wenn Sie den HLS-Simulator auf Ihrem Rechner verwenden wollen, dann müssen Sie einen RabbitMQ-Server installieren. Auf der Webseite des Projektes finden Sie Installer und Installationsanleitungen für alle gängigen Betriebssysteme.

Abgabetermin: entweder am 09.12.2014 oder am 20.01.2015 im Praktikum (bitte geben Sie mir per Mail Bescheid, welchen Termin Sie wählen).

Anlagen
- Vorgegebene Klassen in den Packages `graph, hls, rabbitmq und test`
- Höhenfeld und Farbkarte für Deutschland (*hoehenkarte_deutschland.png, karte_deutschland.jpg*)
- Jar-Bibliotheken für RabbitMQ und den JSON-Parser
- Schnittstellenspezifikation *CgCooperationDocument.pdf*