Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Christian Engwer,
Andreas Buhr M.Sc.

23.04.2014

Übung zum Praktikum

Interaktive Simulationen

SS 2014 — Blatt 2

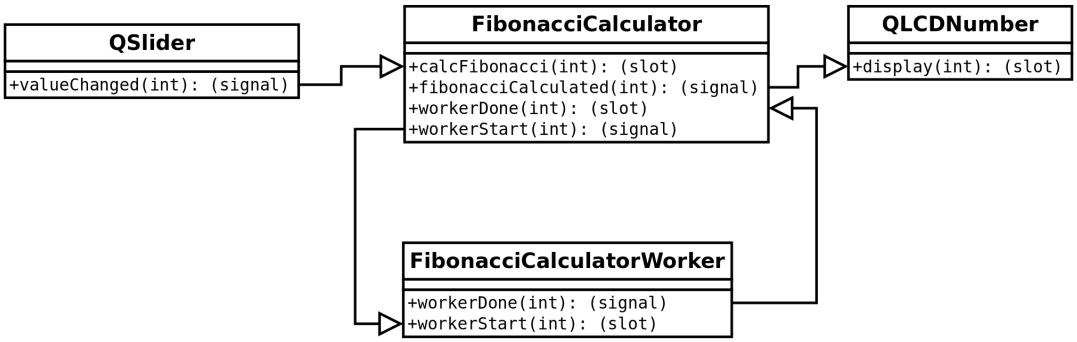
Aufgabe 1 (Multithreading mit Qt)

In dieser Aufgabe "reparieren" Sie einen Fibonaccizahlen-Berechner, der vom User-Interface mit Signalen überschwemmt wird. Selbstverständlich könnte man auch einen besseren Algorithmus zur Berechnung der Fibonaccizahlen einbauen, aber das würde dem Sinn dieser Aufgabenstellung widersprechen.

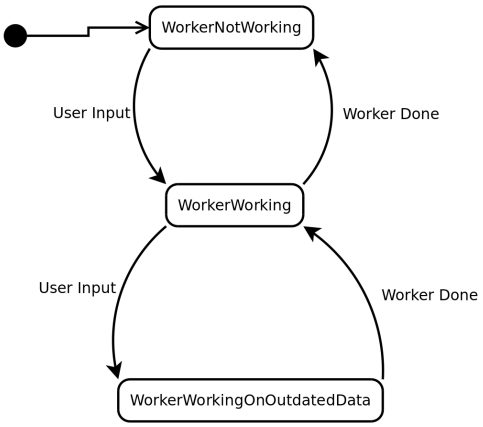
(a) Entpacken Sie das auf der Vorlesungsseite bereitgestellte Fibonacci-Programm und testen Sie es. Überzeugen Sie sich, dass es erwartungsgemäß funktioniert, i.e. ab ca. 40 kaum noch reagiert.

(b) Schicken Sie den FibonacciCalculator in einen eigenen Thread. Überzeugen Sie sich, dass der Slider nun flüssig reagiert, aber keine sinnvolle Ausgabe mehr kommt, da der FibonacciCalculator mehr Signale erhält, als er verarbeiten kann.

(c) Reparieren sie dies, indem Sie den FibonacciCalculator in zwei Klassen aufteilen. Dabei soll der "FibonacciCalculator" die Arbeit koordinieren, während der "FibonacciCalculatorWorker" die Arbeit in einem separaten Thread erledigt.



Der "FibonacciCalculator" soll sich dabei immer den Zustand seines Workers merken und entsprechend handeln. Die Zustände sollen in etwa dem folgenden Diagram entsprechen.



(d) Diese Aufgabe ist erledigt, wenn der Slider jederzeit flüssig reagiert und die Ausgabe der Fibonaccizahl immer in möglichst kurzer Zeit erfolgt.

Aufgabe 2 (Langsame Simulation, langsame Visualisierung, große Visualisierungsdaten)

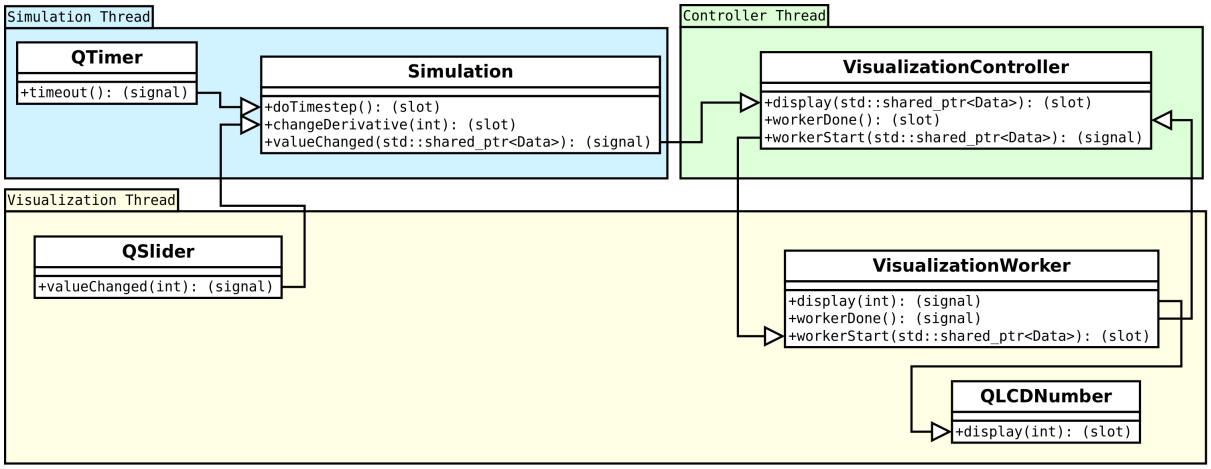
Bei nichttrivialen Simulationen können drei Probleme auftreten:

(a) Langsame Simulation:
Die Simulation selbst kann lange dauern. In dem Fall darf sie nicht im Hauptthread durchgeführt werden, da sonst die grafische Benutzeroberfläche nur langsam reagiert.

(b) Langsame Visualisierung:
Die Visualisierung kann langsamer sein als die Simulation. In diesem Fall erhält die Visualisierung mehr Signale, als sie verarbeiten kann. Dies ist äquivalent zu dem Fall in Aufgabe 1, eine Lösung funktioniert entsprechend.

(c) Große Visualisierungsdaten:
Wenn die Daten zur Visualisierung zu groß werden, sollten die Daten nicht mehr kopiert werden. Es empfiehlt sich, statt dessen Pointer zu verschicken.

Für alle drei Probleme haben Sie in der Vorlesung Lösungen kennen gelernt. Modifizieren Sie Ihr Programm von Aufgabenblatt 1, Aufgabe 2 so, dass diese Probleme nicht auftreten können. Orientieren Sie sich an dem in der Vorlesung vorgestellten Programmdesign:



Testen Sie Ihre Lösung:

(a) Simulieren Sie einen aufwändigen Zeitschritt. Lassen Sie dazu Ihr Programm in der "doTimestep"-Funktion eine Sekunde warten. Prüfen Sie, dass das User-Interface jederzeit flüssig reagiert und Ihre Einstellungen am Slider im nächstmöglichen Zeitschritt Wirkung zeigen. Entfernen Sie die Wartezeit wieder.

(b) Simulieren Sie eine aufwändige Visualisierung. Lassen Sie dazu Ihren "VisualizationWorker" eine Sekunde warten, bevor er den aktuellen Wert anzeigt. Das Userinterface reagiert dann nicht mehr flüssig (das ist nicht so einfach möglich). Aber die Benutzereingaben sollten zum nächstmöglichen Zeitschritt Wirkung zeigen. Entfernen Sie danach die Wartezeit wieder.

(c) Simulieren Sie große Datenmengen. Dazu legen Sie eine Klasse "Data" an, welche einen "int" enthält, aber daneben auch noch ein Array von 100 MB. Benutzen Sie "Data"-Objekte, um Ihren int zu transportieren. Die CPU-Auslastung Ihrer Simulation sollte dadurch nicht merklich ansteigen, da Sie ja nur Pointer verschicken.