Softwareprojekt II bei Prof. Ch. Knabe

Projektteam: Muehle
Dennis Rumpf
Yakup Krahan
Andrzej Kozlowski
Eric Esser
Christian Haupt

Mühle Online ist eine Online Spielplattform bzw. multiuser System in der 2 Spieler gegeneinander Mühle spielen können und via Chat Nachrichten untereinander austauschen können. Des weiteren ist das System auf (Spiele)Services ausgelegt welches es leicht ermöglicht weitere Spiele der Plattform hinzuzufügen. Die Plattform verfügt über einen Administrationsbereich wo der Admin in der Lage ist an Spieler Rechte zu vergeben, sie löschen oder ändern.
Das System wurde mit verschiedenen Techniken realisiert. Im Backend wurde Java eingesetzt und Frontend Flex mit Actionscript 3, die Daten werden mit Hibernate auf einer relationalen MySql Datenbank persistiert.
Besonders große Probleme gab es mit dem Front End und die Logik daraus. Im gegensatz zum Backend oder der Spiellogik ist es hier nicht möglich einfach so ein Ausnahmehandling zu schreiben. Bei einer falschen Berechnung oder einem falschen Ergebnis kann man Fehler ausgeben. Bei der Oberfläche ist es in dem Sinne kein Fehler, wenn z.b. mehrere Schwarze Steine liegen. Man muss alle möglichen Fehlerquellen im Voraus schon bedenken und austesten. Da ist auch schon die zweite große Schwierigkeit. Das manuelle Testen. Es gibt zwar Tools zum testen von GUIs und Oberflächen, uns wäre aber nicht bekannt, dass man ein komplettes Mühlespiel auf Fehler testen kann. Das letzte Problem war die Umrechnung der LGFields in Pixel und umgekehrt. Da haben wir uns ein FieldUtil gebastelt, der das übernimmt. Aus mathematischer und logischer Sicht war es trotzdem nicht ganz einfach die Fälle zu testen, wo mit geringstem Rechenaufwand das Ergebnis raus kommt. Dies in Kombination mit den anderen Fehlern, hat gezeigt, dass man besonders für diesen Teil sehr viel Zeit einräumen muss und bei jeder kleineren Änderung im Back-End auch wieder alles anpassen muss. Wir hätten mit diesem Teil besonders früh anfangen müssen. Am besten noch früher als alles andere. Dies war einer der Gründe, wieso unser Projekt so fehlerbehaftet war. Das war eine wichtige Erkenntnis für die Zukunft.
Das Konzept der Spielkommunikation haben mit dem Freamwork BlazeDS realisiert. BlazeDS bietet einen guten Messaging Service.
Doch die Programmierung war nicht ohne Schwierigkeiten und wärend der realisierung traten Problemen in der Anpassung. Es mussten drei unabhängig entwickelte Teile der Anwendung miteinander kommunizieren und somit angepasst werden.
Unser in Java geschriebenes Backend, besteht aus 3 Ebenen – einer Logikebene, einer Serviceebene und einer Persistierungsebene. Diese Schichttrennung korrekt einzuhalten war zuerst einmal schwierig. Ungewohnt war anfangs die klar abgegrenzte Modularisierung und Ebenentrennung, uns schien, der Code würde unnötig aufgebläht. Es dauerte lange, die erste Transaktion durch die Ebenen zu schreiben. Allerdings waren die darauf folgenden Transaktionen dafür umso einfache zu bewerkstelligen. Als dann einmal die Architektur stand, ging es relativ schnell von statten, die verbleibenden Funktionen zu implementieren, da die eigentliche Arbeit die war, die Schema korrekt umzusetzen.
Eine große Erleichterung stellte die Arbeit mit JUnit dar, allerdings
wurde sich dem erst gegen Ende des Projektes gewahr. Im Gegensatz dazu, wie es angezeigt wäre, JUnit zu benutzen, haben wir die Tests nachträglich gebaut, als der Code schon stand. Diese Arbeit ging zuerst auch schleppend und es schien sich um völlig unnötige Arbeit zu handeln. Als jedoch die Struktur der Tests erst einmal stand, war es ein Leichtes, im Nachhinein hinzugefügte oder

geänderte Funktionen zu überprüfen. Angesichts der Tatsache, dass es ein weiteres Team die Funktionen der Fassaden-Klasse dringend für ihre Tests brauchte, stellte das sicher, dass nicht die Fehlersuche auf der Frontendentwicklung unnötig durch ein fehlerhaftes Backend erschwert wird. Angesichts der Tatsache, dass in einer realen Situation plötzliche Kundenwünsche, späteres Hinzufügen von Features und Erweiterung der Funktionalität plötzlich auftauchen können, erschien uns diese Art des Herangehens sehr sinnig, und unser Code war gut aufgehoben in der engen Struktur der JUnit-Tests.
Wir haben uns für unser System, für die Strategie des Zentrales-Ausnahme-Melden entschieden. Jede Ausnahme, wird von der Schicht in der sie auftritt, bis zu Fassade hoch gereicht. Sie enthalten wichtige Parameter zur Lokalisierung der Fehler und dessen Auslöser. In der Fassade werden sie, über ein Loggingsystem, auf dem Server geloggt/gespeichert, um diese auch noch lange nach Auftreten verfolgen
zu können. Benutzerspeziefische-Ausnahmen werden dann an ihm auf den Flexclient in allgemeinerer Form weitergereicht.