Verteilte Systeme – Mehrspieler - Sudoku

Michael Giger micgiger
Jan Wolf wolfja
Sebastian Wicki swicki
Martina Seps sepsma
Marcel Marti marmarti
Thomas Meier thomeier
Niculin Tschurr tschurrn

ABSTRACT

In diesem Report beschreiben wir unser Mehrspieler-Sudoku. Anders als bei bestehenden Produkten, kann jeder Spieler sein eigenes Android-Gerät benutzen. Die Kontrahenten spielen dasselbe Sudoku und sehen somit die Änderungen des Anderen. Wir beschreiben sowohl die verschiedenen Arbeitsprozesse, als auch das Produkt selbst. Neben einem Einblick in einige Implementationsdetails beschreiben wir, welche Herausforderungen dieses Projekt mit sich brachte und wie wir diese angingen. Unser Produkt ist auf vielfältige Weise erweiterbar, deshalb zeigen wir ebenfalls, wie es in Zukunft verbessert werden könnte.

EINFUEHRUNG

Ziel der Projektaufgabe war es, eine Android-Applikation zu entwickeln, welche auf verschiedenen Geräten läuft, welche miteinander Informationen austauschen. Unser Sudoku erreicht genau dies. Jeder Spieler kann sich über unsere Applikation registrieren und nach anderen Spielern suchen. Es stehen verschiedene Schwierigkeitsstufen und Hilfsmittel, wie automatische Notizen, sogenannte Pencilmarks, zur Verfügung. Im Moment existiert ein zweispieler Modus, wobei jeder Spieler eine Internetverbindung und unsere Applikation auf seinem Androidgerät braucht. Die Idee ist, dass mehrere Spieler gleichzeitig dasselbe Sudoku lösen und dabei jeweils die Lösungen der Gegner sehen und auch korrigieren können. Wer die meisten Felder richtig ausfüllt gewinnt das Spiel.
Wir haben in kleinen Teams einzelne Teile des Projektes implementiert und sehr darauf geachtet, dass unser Projekt möglichst erweiterbar ist. Im Verlauf dieses Reports werden wir unsere Arbeitsweise, technische Details, sowie mögliche Erweiterungen im Detail erläutern.

ARBEITSWEISE

An diesem Projekt haben sieben Personen teilgenommen. Damit alle am selben Code arbeiten konnten, richteten wir ein GitHub-Repository ein. Neben den wöchentlichen Treffen benutzten wir häufig das von GitHub zur Verfügung gestellte Wiki. Wir haben uns in vier Gruppen von jeweils ein oder zwei Personen aufgeteilt, in ein GUI-, Logik-, Server-, und Kommunikationsteam. Diese Aufteilung geschah hauptsächlich, weil diese vier Gebiete ausser einigen Schnittstellen relativ unabhängig entwickelt werden konnten. Es müssen sich nicht immer alle mit allen absprechen. So hat zum Beispiel das Team, welches das GUI entwickelt, wenig Berührung mit dem Code des Serverteams. Um die Schnittstellen zwischen den Teams genau zu definieren haben wir alles in unserem Wiki nachgeführt und den gesamten Code damit im Detail erläutert. Im weiteren Verlauf dieses Dokumentes werden die einzelnen Teams auf ihren Teil des Projektes eingehen.

GUI

Allgemeines

Das GUI besteht aus zwei Activities. Zum einen ist es die MainActivity, welche das Spiel-Setup regelt und die GameActivity, welche das Spiel selber anzeigt.
Die Benutzeroberfläche wurde nach verschiedenen Kriterien gestaltet. Zuerst sollte sie einfach sein, weil die Applikation sowohl für Smartphones als auch Tabletts designed ist. D.h. wir wollten möglichst wenige Buttons oder andere Views, die die Benutzeroberfläche überladen. Als zweiten Punkt wollten wir möglichst wenige Listener/ Backgroundworker, welche zur gleichen Zeit arbeiten, sodass der Stromverbrauch auf ein Minimum reduziert wird.

Setup eines Spiels

In der ersten Activity beim Start der Applikation kann man ein Spiel eröffnen oder suchen. Dazu muss man sich zuerst beim Server registrieren, indem man einen Username eingibt und auf “Register” klickt. Danach kann man einen Schwierigkeitsgrad wählen. Wenn man das gemacht hat, hat man zwei Optionen:

1. Man gibt einen Username eines gewünschten Gegners ein (z.b. jemanden, der neben einem sitzt und den man kennt) und klickt auf “Search”. Ist dieser beim Server registriert erhält er eine Einladung für das Spiel. Auf dem eigenen Gerät erscheint eine runde ProgressBar.

2. Man lässt das Feld leer und klickt auf “Search”. Der Server weist einem dann einen zufälligen Gegner zu, der ebenfalls die zweite Option gewählt hat.

Tastatur

Die Eingabe erfolgt über eine eigens angelegte Tastatur. Sie enthält die Zahlen von 1 bis 9, einen “Insert”-Button, einen “Delete”-Button und den “Pencilmarks”-Button. Über die Zahlen 1 bis 9 kann man die aktive Zahl ändern. Insert, Delete und Pencilmarks beschreiben die Funktionen des User-Inputs. Ist Insert an, dann kann der User Zahlen einfügen. Ist Delete an, kann der User eigene Zahlen löschen und wenn Pencilmarks an ist, können Pencilmarks hinzugefügt oder gelöscht werden.
Es sind immer genau 2 Buttons der Tastatur aktiv. Dies sind

zum einen eine aktive Zahl (1-9) und zum anderen eine der drei Funktionen (Ins, Del, Pen).
Die Tastatur wurde mittels einer GridView implementiert, welche ImageView's beinhaltet. Die Klasse ImageViewAdapter regelt den Inhalt der GridView abhängig vom Userinput. Dazu wurde ein Listener implementiert, welcher abfängt, an welche stelle in das Grid geklickt wurde. Dann werden die nötigen Felder gehighlightet und der Adapter der Tastatur angepasst.
Die Tastaturgrösse ist abhängig von der Breite des Screens, sodass die Tastatur bei einem Tablet grösser erscheint als auf einem Smartphone.

Spielfeld

Das Spielfeld besteht aus 81 Zellen (9x9), welches wir mittels einer GridView implemtiert haben. Diese GridView beinhaltet wiederum TextViews, welche dann die Zahlen bzw. Pencilmarks anzeigen. Dazu haben wir einen onClickListener implementiert, welcher dann Operationen auf den Zellen ausführt.
Zu Beginn wird einmalig im OnCreate ein TextViewAdapter gesetzt, welcher dem GridView die TextViews übergibt.
Ist z.B. die Funktion "Insert" aktiv und man klickt auf das linke obere Feld, dann wird ein insertRequest an die Logik gesendet und schliesslich an den Server weitergeleitet.

SudokuChangeListener

Das GUI implementiert das Inteface SudokuChangeListener, welches die Methode onSudokuChanged(SudokuChangedEvent e) beinhaltet. In dieser Methode werden alle Anpassungen am GUI vorgenommen, welche in Kraft treten, wenn das Sudoku sich ändert (Hinzufügen einer Zahl, Löschen einer Zahl, Ein- und Ausblenden eines Pencilmarks)

Menu

Um Pencilmarks während des Spiels an- bzw. auszuschalten haben wir einen Eintrag ins OptionsMenu gemacht. Dieser togglet die Pencilmarks auf dem Spielfeld und ist über den Button "Menu" von Android erreichbar.

Farben

Die Farbe "Rot" steht in unserer Applikation immer für den Gegner und die Farbe "Blau" für einen selbst. Die Gestaltung des Hintergrunds und der vom User eingefügten Zahlen ist abhängig von verschiedenen Parametern. Eine vom User eingefügte Zahl erscheint immer in sattem Blau, die Zahl des Gegners in sattem Rot. Das leichte rot/rosa im Hintergrund bedeutet, dass der Gegner führt. Wenn der Hintergrund jedoch hellblau ist, dann führt man selbst. Der Score selbst wird also nicht angezeigt. Wir dachten, dass wir so die Spannung für den User erhöhen können.
Gelb hinterlegte Felder sind diejenigen, welche die aktive Zahl beinhalten. Braun hinterlegte Felder sind diejenigen, welche die aktive Zahl als Pencilmarks beinhalten.

Probleme und deren Loesung

1. Wir hatten das Problem, dass bei Android die Tastatur manuell zu jeder Zeit eingeblendet werden kann, wenn man lange auf den Menu-Button klickt.
   Lösung: Wir haben die manuelle Einblendung unterdrückt, indem wir die Methoden onKeyDown, onKeyLongPress und onKeyUp überschrieben haben.

2. Beim klassischen Sudoku-Spielfeld sind die Abgrenzungen zwischen jeder dritten Spalte/Zeile dicker. Die GridView lässt allerdings keine verschiedene Abstände zwischen den Items zu.
   Lösung: Um die 3x3 Boxen trotzdem erkenntlich zu machen, haben wir verschiedene Hintergrundfarben benutzt. (Grau und Weiss)

3. Um die Pencilmarks anzuzeigen wollten wir zuerst eine 3x3 GridView in die Einzelnen Zellen des Spielfelds einfügen. Dies gestaltete sich jedoch sehr umständlich.
   Lösung: Wir benutzten einfach eine dreizeilige TextView um die Pencilmarks anzuzeigen.

LOGIK

Die Aufgabe war es die Logik und die interne Repräsentierung des Sudokus zu implementieren.
Während unserem ersten Treffen, haben wir viele Ideen für den Spielablauf und verschiedenste Features diskutiert. Wir wollten dem Spieler die Möglichkeit geben, Zahlen für sich zu behalten die eventuell auch von den Zahlen auf dem Server abweichen. So sollte ein Spieler die Möglichkeit haben, alle "einfachen" Zahlen zu setzen, die sich aus einer "schwierigen" Zahl ergeben die er gefunden hat. Natürlich mit dem Risiko, dass in der Zwischenzeit ein anderer Spieler das gleiche herausfindet. Erst viel später kam die Erkenntnis, dass es fast unmöglich wäre das irgendwie übersichtlich auf einem Mobile Bildschirm darzustellen.

Aufbau

Die Logik wird mit einem SudokuHandler instanziert. Dieser Handler nimmt über Android Messages sowohl Änderungsrequests vom User aus dem GUI entgegen, als auch vom Server. Der SudokuHandler implementiert das SudokuInfo Interface, dass dem GUI Informationen über den jetzigen Zustand des Spielfelds zur Verfügung stellt. Ausserdem implementiert es das Interface SudokuChangePublisher, an das sich das GUI anmeldet um über Änderungen informiert zu werden. Der SudokuHandler besitzt eine Instanz des Server Interface und kann so Änderungen an den Server schicken.
Ausserdem besitzt er eine Instanz vom SudokuGrid welches das Spielfeld und vorallem auch die Pencilmarks verwaltet.

Herausforderungen

Das Speichern und Setzen von Zahlen war schnell implementiert. Auch das Anpassen von Pencilmarks war relativ simpel. Das eigentliche Problem kommt beim Ändern von Zahlen und der dazugehörigen Korrektur der Pencilmarks.
Das erste Feature dass ich implementiert habe, war die Möglichkeit einen Zug rückgängig zu machen, so dass alle dazugehörigen Pencilmarkänderungen ebenfalls rückgängig gemacht werden. Leider findet dieses Feature keine Anwendung in unserem Spiel, da wir keine lokalen Züge mehr haben, die Rückgängig gemacht werden könnten.
Eine weitere Herausforderung war, die Pencilmarks wieder

herzustellen, nachdem eine Zahl gelöscht oder überschrieben wird. Die zugehörigen Pencilmarks sollen dann wieder auftauchen, aber diejenigen die der Spieler selber eliminiert hat sollen eliminiert bleiben.
Die grösste Herausforderung war, dass die Logik zwischen der Kommunikation und dem GUI angesiedelt ist. Weil diese ja gleichzeitig entwickelt wurden, standen sie nicht zur Verfügung zum Testen.
Wir haben das Problem so gelöst, dass wir für uns selber ein sehr einfaches Mock-Gui in Swing und einen sehr einfachen Mock-Server geschrieben haben. Der Mock-Server war natürlich kein Server, sondern hat nur Nachrichten an zwei Instanzen des Spiels weitergeleitet. Aber so war es relativ einfach zu sehen wie sich das Spiel verhält. Als zusätzlichen Bonus konnte so alles lokal auf einem Computer laufen gelassen werdenund es blieb erspart, jede kleine Änderung auf zwei Android Geräte laden zu müssen.

Zukunft
Wenn wir mehr Zeit hätten, würde ich gerne versuchen alle Züge lokal rückgängig machbar zu implementieren, auch die des Gegners. So wäre es sehr viel einfacher einen Fehler im Spiel zu finden und zu korrigieren.

KOMMUNIKATION
Bei unserem Kommunikations-Layer achteten wir besonders darauf, dass die API für die darüber liegenden Layer so einfach und verständlich wie möglich ist. Dazu verwendeten wir ein einfaches System aus Listener, Client und Server und verstecktem Threading und Networking innerhalb des Layers.

Nachrichtenverarbeitung
Sowohl Server- als auch Client bieten die Möglichkeit an, einen eigenen Handler, der durch das *ServerMessageHandler*-, bzw. *ClientMessageHandler*-Interface beschrieben wird zu setzen. Ist ein Handler gesetzt, werden fortan alle eingehenden Nachrichten an ihn weitergeleitet. Die unterschiedlichen Nachrichtentypen werden direkt durch verschiedene Klassen in Java ausgedrückt. Wir haben also eine sehr breite Vererbungshierarchie an deren Wurzel die einfachste Nachricht, *Message* steht. Die Instanzierung dieser Nachrichten wurde von uns weiter abstrahiert und direkt in den Client- bzw. Server eingebaut. Sollte dennoch die Notwendigkeit bestehen eine Nachricht von Hand aufzubauen, bieten Client und Server auch dazu entsprechende Methoden an.

Nachrichtenformat
Nachrichten werden mit GSON [1] erwähnen. Ein System, einer Bibliothek für Java zur Serialisierung und Deserialisierung beliebiger Objekte in JSON, in einen JSON-String serialisiert und so direkt über Java-Sockets versendet. Wir haben uns für JSON entschieden, weil das Datenformat einfach zu verstehen ist. Die Wahl von JSON hat für uns das Debuggen stark vereinfacht, da wir die Nachrichten so ohne einen weiteren Zwischenschritt ansehen und analysieren konnten.

Threading und Networking
Client, Server und Listener sind intern gethreaded. Jedes Server- und Clientobjekt benutzt intern zwei Threads, einen um ausgehende Nachrichten zu versenden, den anderen, um eingehende Nachrichten zu verarbeiten. Da die eingehenden Nachrichten direkt verarbeitet und an den registrierten Handler weitergeleitet oder gebuffert werden, bis ein Handler registriert wurde, werden die Funktionen des Handlers auch im Empfängerthread aufgerufen. Dieser Umstand musste von den Komponenten, die auf den Kommunikations-Layer aufbauten beachtet werden. Der Listener benutzt einen einzelnen Thread, um neue Verbindungen zu akzeptieren und daraus neue Clients zu erstellen.

Zukunft
Unsere Nachrichten werden zur Zeit noch unverschlüsselt übermittelt. Das ermöglicht jedem Entwickler, einen eigenen Clienten für unser Sudoku zu entwickeln und die Sudokus algorithmisch zu lösen anstatt wirklich zu spielen. Für spätere Versionen und eine eventuelle Veröffentlichung müssten wir das noch anpassen.

SERVER
Die Aufgaben des Servers umfassen das Usermanagement, sowie das Erfassen und Broadcasten der Spielzüge. Auch übernimmt der Server das Scoring, bestimmt also wer jeweils das Spiel anführt und muss am Anfang eines neuen Spiels sowohl Sudokus erzeugen, als auch im weiteren Verlauf die Richtigkeit der Spielereingaben überprüfen. Beim implementieren des Servers haben wir besonders darauf geachtet, dass der Server zukünftige spezialisiertere Versionen unserer Sudokuapplikation unterstützt, ohne dass am grundsätzlichen Design etwas geändert werden muss.

Threading
Alle Nachrichten, welche der Server von verschiedenen Clients erhält werden in einer globale FIFO-Queue aufgenommen. Somit stellen wir die zeitlich richtige Ausführung aller Aktionen sicher. Sobald sich zwei User auf ein Spiel geeinigt haben, wird auf dem Server ein neuer Thread für dieses Spiel gestartet, die Nachrichten der Spieler werden weiterhin in die FIFO-Queue geschrieben, danach aber vom entsprechenden Thread bearbeitet. Dabei wird als erstes überprüft ob der Spielzug gültig bzw. erlaubt ist. Sofern der Zug gültig ist, wird der neue Punktestand entsprechend den unten beschriebenen Regeln angepasst und danach allen Mitspielern mitgeteilt, dass sich das Sudoku verändert hat. Ist das Spiel fertig gelöst, oder verlässt einer der Spieler das aktuelle Spiel, so wird das Spiel beendet und der Thread wird beendet.

Scoring
Hier erleutern wir noch kurz, nach welchen Regeln wir die Punkte vergeben. Zu einem späteren Zeitpunkt, könnten diese Angaben noch in die Applikation integriert werden.
Grundsätzlich bekommt ein Spieler bei jedem Spielzug einen Punkt, auch wenn er einen Fehler macht. Somit stellen wir bei einem Kopf an Kopf Rennen sicher, dass jemand nicht gleich bemerkt, dass die letzte Eingabe ein Fehler war, weil

sich die Eingabe negativ auf den eigenen Punktestand auswirkt. Korrigiert ein Spieler eine falsche gegnerische Zahl, so erhält er zwei Punkte und dem Gegner werden jetzt drei Punkte abgezogen. Löscht ein Spieler seine eigene falsche Zahl, wird ihm ein Punkt abgezogen, er erlitt also keinen Schaden. Wird eine richtige Zahl "korrigiert", so erhält der Spieler zuerst einen Punkt, sofern seine Zahl aber wieder mit der richtigen Lösung überschrieben wird, so werden ihm wie bereits oben beschrieben drei Punkte abgezogen. Das Spiel ist beendet, sobald alle Felder richtig ausgefüllt sind. Der Spieler mit dem höchsten Punktestand gewinnt.

Datenbank

Der Verlauf der aktuellen Spiele wird in einer Datenbank gespeichert, ebenso die Anfangswerte ("Clues") der Sudokus, geordnet nach Schwierigkeitsgrad, und die Lösungen zu allen Spielen. Bei der Spielform, wie sie jetzt besteht wäre eine Datenbank nicht nötig, es könnte alles in Java mit Arrays oder Listen implementiert werden. Wir entschieden uns trotzdem für eine Datenbank, weil wir so ohne grosse Anpassungen am Server viele Erweiterungsmöglichkeiten haben, die zum Teil weiter unten erläutert werden.
Die Lösungen und Clues haben wir mithilfe eines Open-Source-Tools [2] erstellt und danach mit einem Python-Skript zu unserer Datenbank hinzugefügt.
Wir haben uns für MongoDB [3] entschieden, da diese Datenbank OpenSource und durch ihre Schemafreiheit einfach zu erweitern ist. Die Datenbank verwaltet JSON ähnliche Dokumente, sogenannte "Collections". Ausserdem interessierten wir uns für eine SQL-Alternative und MongoDB ist eine der bekanntesten.

Zukunft

Da bei der Implementation des Servers viel Wert auf Erweiterbarkeit gelegt wurde, wäre es ohne grossen Aufwand möglich, dass Spieler sich langfristig registrieren könnten, und sich mit einem Passwort einloggen müssen. Somit wäre es möglich, dass ein Spieler ein Spiel verlassen kann und zu einem späteren Zeitpunkt weiterspielen kann, so zB. auch wenn er für kurze Zeit die Internetverbindung verlieren sollte. Auch könnte ein Spieler an verschiedenen Spielen gleichzeitig teilnehmen. Eine andere Erweiterung wäre ein Modus mit beliebig vielen Teilnehmern pro Spiel, anstatt wie bisher nur zwei.

FAZIT

Die Herausforderungen bei diesem Projekt bestanden für uns nicht nur bei der Implementierung eines komplexen Programmes, sondern auch bei der Kommunikation im Team, da wir eine relativ grosse Gruppe waren. Durch die Aufteilung in Teilprojekte, die wöchentlichen Meetings und das ausführlich geführte Wiki erreichten wir jedoch effizientes Arbeiten. Entstanden ist ein Mehrspieler-Sudoku, das bereits grossen Spass macht und mit wenig Aufwand auf vielfältige Art und Weise erweitert werden könnte.

REFERENZEN

[1] google-gson, a Java library to convert JSON to Java Objects and vice-versa, http://code.google.com/p/google-gson/
[2]Debian Sudoku Paket, http://packages.debian.org/squeeze/sudoku/
[3] MongoDB, http://www.mongodb.org